SIGNAL PROCESSING WITH SCILAB

Scilab Group

INRIA Meta2 Project/ENPC Cergrene

INRIA - Unité de recherche de Rocquencourt - Projet Meta2
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
E-mail : scilab@inria.fr

Acknowledgement

This document is an updated version of a primary work by Carey Bunks, François Delebecque, Georges Le Vey and Serge Steer

Contents

Chapter 1

Description of the Basic Tools

1.1 Introduction

The purpose of this document is to illustrate the use of the Scilab software package in a signal processing context. We have gathered a collection of signal processing algorithms which have been implemented as Scilab functions.

This manual is in part a pedagogical tool concerning the study of signal processing and in part a practical guide to using the signal processing tools available in Scilab. For those who are already well versed in the study of signal processing the tutorial parts of the manual will be of less interest.

For each signal processing tool available in the signal processing toolbox there is a tutorial section in the manual explaining the methodology behind the technique. This section is followed by a section which describes the use of a function designed to accomplish the signal processing described in the preceding sections. At this point the reader is encouraged to launch a Scilab session and to consult the on-line help related to the function in order to get the precise and complete description (syntax, description of its functionality, examples and related functions). This section is in turn followed by an examples section demonstrating the use of the function. In general, the example section illustrates more clearly than the syntax section how to use the different modes of the function.

In this manual the `typewriter-face` font is used to indicate either a function name or an example dialogue which occurs in Scilab.

Each signal processing subject is illustrated by examples and figures which were demonstrated using Scilab. To further assist the user, there exists for each example and figure an executable file which recreates the example or figure. To execute an example or figure one uses the following Scilab command

```
-->exec('file.name')
```

which causes Scilab to execute all the Scilab commands contained in the file called `file.name`.

To know what signal processing tools are available in Scilab one would type

```
-->disp(siglib)
```

which produces a list of all the signal processing functions available in the signal processing library.

1.2 Signals

For signal processing the first point to know is how to load and save signals or only small portions of lengthy signals that are to be used or are to be generated by Scilab. Finally, the generation of synthetic (random) signals is an important tool in the development in implementation of signal processing tools. This section addresses all of these topics.

1.2.1 Saving, Loading, Reading, and Writing Files

Signals and variables which have been processed or created in the Scilab environment can be saved in files written directly by Scilab. The syntax for the `save` primitive is

```
-->save(file_name[,var_list])
```

where `file_name` is the file to be written to and `var_list` is the list of variables to be written. The inverse to the operation `save` is accomplished by the primitive `load` which has the syntax

```
-->load(file_name[,var_list])
```

where the argument list is identical that used in `save`.

Although the commands `save` and `load` are convenient, one has much more control over the transfer of data between files and Scilab by using the commands `read` and `write`. These two commands work similarly to the read and write commands found in Fortran. The syntax of these two commands is as follows. The syntax for `write` is

```
-->write(file,x[,form])
```

The second argument, `x`, is a matrix of values which are to be written to the file.

The syntax for `read` is

```
-->x=read(file,m,n[,form])
```

The arguments `m` and `n` are the row and column dimensions of the resulting data matrix `x`. and `form` is again the format specification statement.

In order to illustrate the use of the on-line help for reading this manual we give the result of the Scilab command

```
-->help read

read(1)         Scilab Function        read(1)


NAME
  read - matrices read

CALLING SEQUENCE
  [x]=read(file-name,m,n,[format])
  [x]=read(file-name,m,n,k,format)

PARAMETERS

  file-name : string or integer (logical unit number)

  m, n      : integers (dimensions of the matrix x). Set m=-1 if you dont
    know the numbers of rows, so the whole file is read.
  format    : string (fortran format). If format='(a)' then read reads a vec-
    tor of strings n must be equal to 1.

  k     : integer

DESCRIPTION
  reads row after row the mxn matrix x (n=1 for character chain) in the file
  file-name (string or integer).

  Two examples for format are : (1x,e10.3,5x,3(f3.0)),(10x,a20) ( the default
  value is *).

  The type of the result will depend on the specified form. If form is
  numeric (d,e,f,g) the matrix will be a scalar matrix and if form contains
  the character a the matrix will be a matrix of character strings.

  A direct access file can be used if using the parameter k which is is the
  vector of record numbers to be read (one record per row), thus m must be
  m=prod(size(k)).

  To read on the keyboard use read(%io(1),...).

EXAMPLE
  A=rand(3,5); write('foo',A);
  B=read('foo',3,5)
  B=read('foo',-1,5)
  read(%io(1),1,1,'(a)')  // waits for user's input

SEE ALSO
  file, readb, write, %io, x_dialog
```

1.2.2 Simulation of Random Signals

The creation of synthetic signals can be accomplished using the Scilab function `rand` which generates random numbers. The user can generate a sequence of random numbers, a random matrix with the uniform or the gaussian probability laws. A seed is possible to re-create the same pseudo-random sequences.

Often it is of interest in signal processing to generate normally distributed random variables with a certain mean and covariance structure. This can be accomplished by using the standard normal random numbers generated by `rand` and subsequently modifying them by performing certain linear numeric operations. For example, to obtain a random vector y which is distributed N(m_y,Λ_y) from a random vector x which is distributed standard normal (i.e. N(0,I)) one would perform the following operation

$$y = \Lambda_y^{1/2} x + m_y \tag{1.1}$$

where $\Lambda_y^{1/2}$ is the matrix square root of Λ_y. A matrix square root can be obtained using the `chol` primitive as follows taking note that it is the transpose of the matrix obtained from `chol` that is used for the square root of the desired covariance matrix. Sequences of random numbers following a specific normally distributed probability law can also be obtained by filtering. That is, a white standard normal sequence of random numbers is passed through a linear filter to obtain a normal sequence with a specific spectrum. For a filter which has a discrete Fourier transform $H(w)$ the resulting filtered sequence will have a spectrum $S(w) = |H(w)|^2$. More on filtering is discussed in Section 1.8.

1.3 Polynomials and System Transfer Functions

Polynomials, matrix polynomials and transfer matrices are also defined and Scilab permits the definition and manipulation of these objects in a natural, symbolic fashion. Polynomials are easily created and manipulated. The `poly` primitive in Scilab can be used to specify the coefficients of a polynomial or the roots of a polynomial.

A very useful companion to the `poly` primitive is the `roots` primitive. The roots of a polynomial `q` are given by :

```
-->a=roots(q);
```

The following examples should clarify the use of the `poly` and `roots` primitives.

Notice that the first polynomial `q1` uses the `'roots'` default and, consequently, the polynomial takes the form $(s-1)(s-2) = 2 - 3s + s^2$. The second polynomial `q2` is defined by its coefficients given by the elements of the vector. Finally, the third polynomial `q3` calculates the characteristic polynomial of the matrix `a` which is by definition $\det(sI - \mathtt{a})$. Here the calculation of the `roots` primitive yields the eigenvalues of the matrix `a`.

Scilab can manipulate polynomials in the same manner as other mathematical objects such as scalars, vectors, and matrices. That is, polynomials can be added, subtracted, multiplied, and divided by other polynomials. The following Scilab session illustrates operations between polynomials

Notice that in the above session we started by defining a basic polynomial element `x` (which should not be confused with the character string `'x'` which represents the formal variable of the polynomial). Another point which is very important in what is to follow is that division of polynomials creates a rational polynomial which is represented by a list (see `help list` and `help type` in Scilab).

A rational is represented by a list containing four elements. The first element takes the value `'r'` indicating that this list represents a rational polynomial. The second and third elements of the list are the numerator and denominator polynomials, respectively, of the rational. The final element of the list is either `[]` or a character string (More on this subject is addressed later in this chapter (see Section 1.3.2). The following dialogue illustrates the elements of a list representing a rational polynomial.

1.3.1 Evaluation of Polynomials

A very important operation on polynomials is their evaluation at specific points. For example, perhaps it is desired to know the value the polynomial $x^2 + 3x - 5$ takes at the point $x = 17.2$. Evaluation of polynomials is accomplished using the primitive `freq`. The syntax of `freq` is as follows

```
-->pv=freq(num,den,v)
```

The argument `v` is a vector of values at which the evaluation is needed.

For signal processing purposes, the evaluation of frequency response of filters and system transfer functions is a common use of `freq`. For example, a discrete filter can be evaluated on the unit circle in the z-plane as follows

Here, `h` is an FIR filter of length 9 with a triangular impulse response. The transfer function of the filter is obtained by forming a polynomial which represents the z-transform of the filter. This is followed by evaluating the polynomial at the points $\exp(2\pi in)$ for $n = 0, 1, \ldots, 10$ which amounts to evaluating the z-transform on the unit circle at ten equally spaced points in the range of angles $[0, \pi]$.

1.3.2 Representation of Transfer Functions

Signal processing makes use of rational polynomials to describe signal and system transfer functions. These transfer functions can represent continuous time signals or systems or discrete time signals or systems. Furthermore, discrete signals or systems can be related to continuous signals or systems by sampling.

The function which processes a rational polynomial so that it can be represented as a transfer function is called `syslin`:

```
-->sl=syslin(domain,num,den)
```

Another use for the function `syslin` for state-space descriptions of linear systems is described in the following section.

1.4 State Space Representation

The classical state-space description of a continuous time linear system is :

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \\ x(0) &= x_0 \end{aligned}$$

where A, B, C, and D are matrices and x_0 is a vector and for a discrete time system takes the form

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) \\ y(n) &= Cx(n) + Du(n) \\ x(0) &= x_0 \end{aligned}$$

State-space descriptions of systems in Scilab use the `syslin` function :

```
-->sl=syslin(domain,a,b,c [,d[,x0]])
```

The returned value of `sl` is a list where `s=list('lss',a,b,c,d,x0,domain)`.

The value of having a symbolic object which represents a state-space description of a system is that functions can be created which operate on the system. For example, one can combine two systems in parallel or in cascade, transform them from state-space descriptions into transfer function descriptions and vice versa, and obtain discretized versions of continuous time systems and vice versa. The topics and others are discussed in the ensuing sections.

1.5 Changing System Representation

Sometimes linear systems are described by their transfer function and sometimes by their state equations. In the event where it is desirable to change the representation of a linear system there exists two Scilab functions which are available for this task. The first function `tf2ss` converts systems described by a transfer function to a system described by state space representation. The second function `ss2tf` works in the opposite sense.

The syntax of `tf2ss` is as follows

```
-->sl=tf2ss(h)
```

An important detail is that the transfer function `h` must be of minimum phase. That is, the denominator polynomial must be of equal or higher order than that of the numerator polynomial.

```
-->h=ss2tf(sl)
```

The following example illustrates the use of these two functions.

Here the transfer function of a discrete IIR filter is created using the function `iir` (see Section 4.2). The fourth element of `h1` is set using the function `syslin` and then using `tf2ss` the state-space representation is obtained in list form.

1.6 Interconnecting systems

Linear systems created in the Scilab environment can be interconnected in cascade or in parallel. There are four possible ways to interconnect systems illustrated in Figure 1.1. In the figure the symbols s_1 and s_2 represent two linear systems which could be represented in Scilab by transfer function or state-space representations. For each of the four block diagrams in Figure 1.1 the Scilab command which makes the illustrated interconnection is shown to the left of the diagram in typewriter-face font format.

1.7 Discretizing Continuous Systems

A continuous-time linear system represented in Scilab by its state-space or transfer function description can be converted into a discrete-time state-space or transfer function representation by using the function `dscr`.

Consider for example an input-output mapping which is given in state space form as:

$$(C)\left\{\begin{array}{lcl}\dot{x}(t) & = & Ax(t)+Bu(t)\\ y(t) & = & Cx(t)+Du(t)\end{array}\right. \tag{1.2}$$

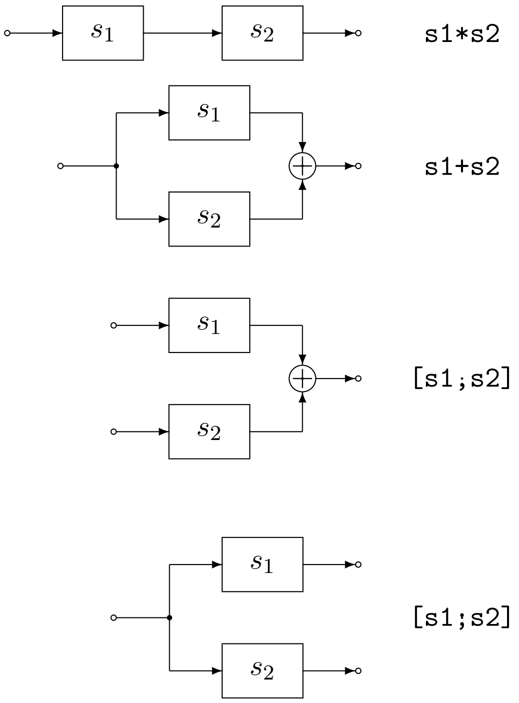

Figure 1.1: Block Diagrams of System Interconnections

From the variation of constants formula the value of the state $x(t)$ can be calculated at any time t as

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\sigma)}Bu(\sigma)d\sigma \tag{1.3}$$

Let h be a time step and consider an input u which is constant in intervals of length h. Then associated with (1.2) is the following discrete time model obtained by using the variation of constants formula in (1.3),

$$(D)\left\{\begin{array}{rcl} x(n+1) & = & A_h x(n) + B_h u(n) \\ y(n) & = & C_h x(n) + D_h u(n) \end{array}\right. \tag{1.4}$$

where

$$A_h = \exp(Ah)$$

$$B_h = \int_0^h e^{A(h-\sigma)}Bd\sigma$$

$$C_h = C$$

$$D_h = D$$

Since the computation of a matrix exponent can be calculated using the Scilab primitive `exp`, it is straightforward to implement these formulas, although the numerical calculations needed to compute $\exp(A_h)$ are rather involved ([30]).

If we take

$$G = \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix}$$

where the dimensions of the zero matrices are chosen so that G is square then we obtain

$$\exp(Gh) = \begin{pmatrix} A_h & B_h \\ 0 & I \end{pmatrix}$$

When A is nonsingular we also have that

$$B_h = A^{-1}(A_h - I)B.$$

This is exactly what the function `dscr` does to discretize a continuous-time linear system in state-space form.

The function `dscr` can operate on system matrices, linear system descriptions in state-space form, and linear system descriptions in transfer function form. The syntax using system matrices is as follows

```
-->[f,g[,r]]=dscr(syslin('c',a,b,[],[]),dt [,m])
```

where `a` and `b` are the two matrices associated to the continuous-time state-space description

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1.5}$$

and `f` and `g` are the resulting matrices for a discrete time system

$$x(n+1) = Fx(n) + Gu(n) \tag{1.6}$$

where the sampling period is `dt`. In the case where the fourth argument `m` is given, the continuous time system is assumed to have a stochastic input so that now the continuous-time equation is

$$\dot{x}(t) = Ax(t) + Bu(t) + w(t) \tag{1.7}$$

where $w(t)$ is a white, zero-mean, Gaussian random process of covariance `m` and now the resulting discrete-time equation is

$$x(n+1) = Fx(n) + Gu(n) + q(n) \tag{1.8}$$

where $q(n)$ is a white, zero-mean, Gaussian random sequence of covariance `r`.

The `dscr` function syntax when the argument is a linear system in state-space form is

```
-->[sld[,r]]=dscr(sl,dt[,m])
```

where `sl` and `sld` are lists representing continuous and discrete linear systems representations, respectively. Here `m` and `r` are the same as for the first function syntax. In the case where the function argument is a linear system in transfer function form the syntax takes the form

```
-->[hd]=dscr(h,dt)
```

where now `h` and `hd` are transfer function descriptions of the continuous and discrete systems, respectively. The transfer function syntax does not allow the representation of a stochastic system.

As an example of the use of `dscr` consider the following Scilab session.

1.8 Filtering of Signals

Filtering of signals by linear systems (or computing the time response of a system) is done by the function `flts` which has two formats . The first format calculates the filter output by recursion and the second format calculates the filter output by transform. The function syntaxes are as follows. The syntax of `flts` is

```
-->[y[,x]]=flts(u,sl[,x0])
```

for the case of a linear system represented by its state-space description (see Section 1.4) and

```
-->y=flts(u,h[,past])
```

for a linear system represented by its transfer function.

In general the second format is much faster than the first format. However, the first format also yields the evolution of the state. An example of the use of `flts` using the second format is illustrated below.

Notice that in the above example that a signal consisting of the sum of two sinusoids of different frequencies is filtered by a low-pass filter. The cut-off frequency of the filter is such that after filtering only one of the two sinusoids remains. Figure **??** illustrates the original sum of sinusoids and Figure **??** illustrates the filtered signal. `exec('flts1.code')` Sum of Two Sinusoidsf.flts1

`exec('flts2.code')` Filtered Signalf.flts2

1.9 Plotting Signals

Here we describe some of the features of the simplest plotting command. A more complete description of the graphics features are given in the on-line help.

Here we present several examples to illustrate how to construct some types of plots.

To illustrate how an impulse response of an FIR filter could be plotted we present the following Scilab session.

Here a band-pass filter with cut-off frequencies of .2 and .25 is constructed using a Hamming window. The filter length is 55. More on how to make FIR filters can be found in Chapter 3. `exec('plot1.code')` Plot of Filter Impulse Responsef.plot1 The resulting plot is shown in Figure **??**.

The frequency response of signals and systems requires evaluating the s-transform on the $j\omega$-axis or the z-transform on the unit circle. An example of evaluating the magnitude of the frequency response of a continuous-time system is as follows.

Here we make an analog low-pass filter using the functions `analpf` (see Chapter 4 for more details). The filter is a type I Chebyshev of order 4 where the cut-off frequency is 5 Hertz. The primitive `freq` (see Section 1.3.1) evaluates the transfer function `hs` at the values of `fr` on the $j\omega$-axis. The result is shown in Figure **??** `exec('plot2.code')` Plot of Continuous Filter Magnitude Responsef.plot2

A similar type of procedure can be effected to plot the magnitude response of discrete filters where the evaluation of the transfer function is done on the unit circle in the z-plane by using the function `frmag`.

```
-->[xm,fr]=frmag(num[,den],npts)
```

The returned arguments are `xm`, the magnitude response at the values in `fr`, which contains the normalized discrete frequency values in the range $[0, 0.5]$.

Here an FIR band-pass filter is created using the function `eqfir` (see Chapter 3). `exec('plot3.code')` Plot of Discrete Filter Magnitude Response f.plot3

Other specific plotting functions are `bode` for the Bode plot of rational system transfer functions (see Section 2.1.1), `group` for the group delay (see Section 2.1.2) and `plzr` for the poles-zeros plot.

Here a fourth order, low-pass, IIR filter is created using the function `iir` (see Section 4.2). The resulting pole-zero plot is illustrated in Figure **??** `exec('plot4.code')` Plot of Poles and Zeros of IIR Filterf.plot4

1.10 Development of Signal Processing Tools

Of course any user can write its own functions like those illustrated in the previous sections. The simplest way is to write a file with a special format . This file is executed with two Scilab primitives `getf` and `exec`. The complete description of such functionalities is given in the reference manual and the on-line help. These functionalities correspond to the button `File Operations`.

Chapter 2

Time and Frequency Representation of Signals

2.1 Frequency Response

2.1.1 Bode Plots

The Bode plot is used to plot the phase and log-magnitude response of functions of a single complex variable. The log-scale characteristics of the Bode plot permitted a rapid, "back-of-the-envelope" calculation of a system's magnitude and phase response. In the following discussion of Bode plots we consider only real, causal systems. Consequently, any poles and zeros of the system occur in complex conjugate pairs (or are strictly real) and the poles are all located in the left-half s-plane.

For $H(s)$ a transfer function of the complex variable s, the log-magnitude of $H(s)$ is defined by

$$M(\omega) = 20\log_{10}|H(s)_{s=j\omega}| \tag{2.1}$$

and the phase of $H(s)$ is defined by

$$\Theta(\omega) = \tan^{-1}[\frac{Im(H(s)_{s=j\omega})}{Re(H(s)_{s=j\omega})}] \tag{2.2}$$

The magnitude, $M(\omega)$, is plotted on a log-linear scale where the independent axis is marked in decades (sometimes in octaves) of degrees or radians and the dependent axis is marked in decibels. The phase, $\Theta(\omega)$, is also plotted on a log-linear scale where, again, the independent axis is marked as is the magnitude plot and the dependent axis is marked in degrees (and sometimes radians).

When $H(s)$ is a rational polynomial it can be expressed as

$$H(s) = C\frac{\prod_{n=1}^{N}(s-a_n)}{\prod_{m=1}^{M}(s-b_m)} \tag{2.3}$$

where the a_n and b_m are real or complex constants representing the zeros and poles, respectively, of $H(s)$, and C is a real scale factor. For the moment let us assume that the a_n and b_m are strictly real. Evaluating (2.3) on the $j\omega$-axis we obtain

$$\begin{aligned} H(j\omega) &= C\frac{\prod_{n=1}^{N}(j\omega-a_n)}{\prod_{m=1}^{M}(j\omega-b_m)} \\ &= C\frac{\prod_{n=1}^{N}\sqrt{\omega^2+a_n^2}e^{j\tan^{-1}\omega/(-a_n)}}{\prod_{m=1}^{M}\sqrt{\omega^2+b_m^2}e^{j\tan^{-1}\omega/(-b_m)}} \end{aligned} \tag{2.4}$$

and for the log-magnitude and phase response

$$M(\omega) = 20(\log_{10} C + (\sum_{n=1}^{N} \log_{10} \sqrt{\omega^2 + a_n^2} - \sum_{m=1}^{M} \log_{10} \sqrt{\omega^2 + b_m^2} \tag{2.5}$$

and

$$\Theta(\omega) = \sum_{n=1}^{N} \tan^{-1}(\omega/(-a_n)) - \sum_{m=1}^{M} \tan^{-1}(\omega/(-b_m)). \tag{2.6}$$

To see how the Bode plot is constructed assume that both (2.5) and (2.6) consist of single terms corresponding to a pole of $H(s)$. Consequently, the magnitude and phase become

$$M(\omega) = -20 \log \sqrt{\omega^2 + a^2} \tag{2.7}$$

and

$$\Theta(\omega) = -j \tan^{-1}(\omega/(-a)). \tag{2.8}$$

We plot the magnitude in (2.7) using two straight line approximations. That is, for $|\omega| \ll |a|$ we have that $M(\omega) \approx -20 \log |a|$ which is a constant (i.e., a straight line with zero slope). For $|\omega| \gg |a|$ we have that $M(\omega) \approx -20 \log |\omega|$ which is a straight line on a log scale which has a slope of -20 db/decade. The intersection of these two straight lines is at $w = a$. Figure **??** illustrates these two straight line approximations for $a = 10$. `exec('bode1.code')` Log-Magnitude Plot of $H(s) = 1/(s - a)$f.bode1

When $\omega = a$ we have that $M(\omega) = -20 \log \sqrt{2}a = -20 \log a - 20 \log \sqrt{2}$. Since $20 \log \sqrt{2} = 3.0$ we have that at $\omega = a$ the correction to the straight line approximation is -3db. Figure **??** illustrates the true magnitude response of $H(s) = (s - a)^{-1}$ for $a = 10$ and it can be seen that the straight line approximations with the 3db correction at $\omega = a$ yields very satisfactory results. The phase in (2.8) can also be approximated. For $\omega \ll a$ we have $\Theta(\omega) \approx 0$ and for $\omega \gg a$ we have $\Theta(\omega) \approx -90°$. At $\omega = a$ we have $\Theta(\omega) = -45°$. Figure **??** illustrates the straight line approximation to $\Theta(\omega)$ as well as the actual phase response. `exec('bode2.code')` Phase Plot of $H(s) = 1/(s - a)$f.bode2

In the case where the poles and zeros of $H(s)$ are not all real but occur in conjugate pairs (which is always the case for real systems) we must consider the term

$$\begin{aligned} H(s) &= \frac{1}{[s - (a + jb)][s - (a - jb)]} \\ &= \frac{1}{s^2 - 2as + (a^2 + b^2)} \end{aligned} \tag{2.9}$$

where a and b are real. Evaluating (2.9) for $s = j\omega$ yields

$$\begin{aligned} H(s) &= \frac{1}{(a^2 + b^2 - \omega^2) - 2aj\omega} \\ &= \frac{1}{\sqrt{\omega^4 + 2(a^2 - b^2)\omega^2 + (a^2 + b^2)} \exp(j \tan^{-1}[\frac{-2a\omega}{a^2+b^2-\omega^2}])}. \end{aligned} \tag{2.10}$$

For ω very small, the magnitude component in (2.10) is approximately $1/(a^2 + b^2)$ and for ω very large the magnitude becomes approximately $1/\omega^2$. Consequently, for small ω the magnitude response can be approximated by the straight line $M(\omega) \approx -20 \log_{10} |a^2 + b^2|$ and for ω large we have $M(\omega) \approx -20 \log |\omega^2|$ which is a straight line with a slope of -40db/decade. These two straight lines intersect at $\omega = \sqrt{a^2+b^2}$. Figure **??** illustrates `exec('bode3.code')` Log-Magnitude Plot of $H(s) = (s^2 - 2as + (a^2+b^2))^{-1}$ f.bode3

the straight line approximations for $a = 10$ and $b = 25$. The behavior of the magnitude plot when ω is neither small nor large with respect to a and b depends on whether b is greater than a or not. In the case where b is less than a, the magnitude plot is similar to the case where the roots of the transfer function are strictly real, and consequently, the magnitude varies monotonically between the two straight line approximations shown in Figure **??**. The correction at $\omega = \sqrt{a^2+b^2}$ is -6db plus $-20\log|a/(\sqrt{a^2+b^2})|$. For b greater than a, however, the term in (2.10) exhibits resonance. This resonance is manifested as a local maximum of the magnitude response which occurs at $\omega = \sqrt{b^2-a^2}$. The value of the magnitude response at this maximum is $-20\log|2ab|$. The effect of resonance is illustrated in Figure **??** as the upper dotted curve. Non-resonant behavior is illustrated in Figure **??** by the lower dotted curve.

The phase curve for the expression in (2.10) is approximated as follows. For ω very small the imaginary component of (2.10) is small and the real part is non-zero. Thus, the phase is approximately zero. For ω very large the real part of (2.10) dominates the imaginary part and, consequently, the phase is approximately $-180°$. At $\omega = \sqrt{a^2+b^2}$ the real part of (2.10) is zero and the imaginary part is negative so that the phase is exactly $-90°$. The phase curve is shown in Figure **??**. `exec('bode4.code')` Phase Plot of $H(s) = (s^2 - 2as + (a^2+b^2))^{-1}$ f.bode4

How to Use the Function `bode`

The description of the transfer function can take two forms: a rational polynomial or a state-space description .

For a transfer function given by a polynomial `h` the syntax of the call to `bode` is as follows

```
-->bode(h,fmin,fmax[,step][,comments])
```

When using a state-space system representation `sl` of the transfer function the syntax of the call to `bode` is as follows

```
-->bode(sl,fmin,fmax[,pas][,comments])
```

where

```
-->sl=syslin(domain,a,b,c[,d][,x0])
```

The continuous time state-space system assumes the following form

$$\begin{aligned} \dot{x}(t) &= \mathtt{a}x(t) + \mathtt{b}u(t) \\ y(t) &= \mathtt{c}x(t) + \mathtt{d}w(t) \end{aligned}$$

and `x0` is the initial condition. The discrete time system takes the form

$$\begin{aligned} x(n+1) &= \mathtt{a}x(n) + \mathtt{b}u(n) \\ y(n) &= \mathtt{c}x(n) + \mathtt{d}w(n) \end{aligned}$$

Examples Using `bode`

Here are presented examples illustrating the state-space description, the rational polynomial case. These two previous systems connected in series forms the third example.

In the first example, the system is defined by the state-space description below

$$\begin{aligned}\dot{x} &= -2\pi x + u\\ y &= 18\pi x + u.\end{aligned}$$

The initial condition is not important since the Bode plot is of the steady state behavior of the system.

The result of the call to `bode` for this example is illustrated in Figure ??. `exec('bode5.code')` Bode Plot of State-Space System Representation f.bode5

The following example illustrates the use of the `bode` function when the user has an explicit rational polynomial representation of the system.

The result of the call to `bode` for this example is illustrated in Figure ??. `exec('bode6.code')` Bode Plot of Rational Polynomial System Representation f.bode6

The final example combines the systems used in the two previous examples by attaching them together in series. The state-space description is converted to a rational polynomial description using the `ss2tf` function.

Notice that the rational polynomial which results from the call to the function `ss2tf` automatically has its fourth argument set to the value `'c'`. The result of the call to `bode` for this example is illustrated in Figure ??. `exec('bode7.code')` Bode Plot Combined Systems f.bode7

2.1.2 Phase and Group Delay

In the theory of narrow band filtering there are two parameters which characterize the effect that band pass filters have on narrow band signals: the phase delay and the group delay.

Let $H(\omega)$ denote the Fourier transform of a system

$$H(\omega) = A(\omega)e^{j\theta(\omega)} \tag{2.11}$$

where $A(\omega)$ is the magnitude of $H(\omega)$ and $\theta(\omega)$ is the phase of $H(\omega)$. Then the phase delay, $t_p(\omega)$, and the group delay, $t_g(\omega)$, are defined by

$$t_p(\omega) = \theta(\omega)/\omega \tag{2.12}$$

and

$$t_g(\omega) = d\theta(\omega)/d\omega. \tag{2.13}$$

Now assume that $H(\omega)$ represents an ideal band pass filter. By ideal we mean that the magnitude of $H(\omega)$ is a non-zero constant for $\omega_0 - \omega_c < |\omega| < \omega_0 + \omega_c$ and zero otherwise, and that the phase of $H(\omega)$ is linear plus a constant in these regions. Furthermore, the impulse response of $H(\omega)$ is real. Consequently, the magnitude of $H(\omega)$ has even symmetry and the phase of $H(\omega)$ has odd symmetry.

Since the phase of $H(\omega)$ is linear plus a constant it can be expressed as

$$\theta(\omega) = \left\{\begin{array}{ll} \theta(\omega_0) + \theta'(\omega_0)(\omega - \omega_0), & \omega > 0\\ -\theta(\omega_0) + \theta'(\omega_0)(\omega + \omega_0), & \omega < 0\end{array}\right. \tag{2.14}$$

where ω_0 represents the center frequency of the band pass filter. The possible discontinuity of the phase at $\omega = 0$ is necessary due to the fact that $\theta(\omega)$ must be an odd function. The expression in (2.14) can be rewritten using the definitions for phase and group delay in (2.12) and (2.13). This yields

$$\theta(\omega) = \begin{cases} \omega_0 t_p + (\omega - \omega_0)t_g, & \omega > 0 \\ -\omega_0 t_p + (\omega + \omega_0)t_g, & \omega < 0 \end{cases} \tag{2.15}$$

where, now, we take $t_p = t_p(\omega_0)$ and $t_g = t_g(\omega_0)$.

Now assume that a signal, $f(t)$, is to be filtered by $H(\omega)$ where $f(t)$ is composed of a modulated band-limited signal. That is,

$$f(t) = f_l(t)\cos(\omega_0 t) \tag{2.16}$$

where ω_0 is the center frequency of the band pass filter and $F_l(\omega)$ is the Fourier transform a the bandlimited signal $f_l(t)$ ($F_l(\omega) = 0$ for $|\omega| > \omega_c$). It is now shown that the output of the filter due to the input in (2.16) takes the following form

$$g(t) = f_l(t + t_g)\cos[\omega_0(t + t_p)]. \tag{2.17}$$

To demonstrate the validity of (2.17) the Fourier transform of the input in (2.16) is written as

$$F(\omega) = \frac{1}{2}[F_l(\omega - \omega_0) + F_l(\omega + \omega_0)] \tag{2.18}$$

where (2.18) represents the convolution of $F_l(\omega)$ with the Fourier transform of $\cos(\omega_0 t)$. The Fourier transform of the filter, $H(\omega)$, can be written

$$H(\omega) = \begin{cases} e^{\omega_0 t_p + (\omega - \omega_0)t_g}, & \omega_0 - \omega_c < \omega < \omega_0 + \omega_c \\ e^{-\omega_0 t_p + (\omega + \omega_0)t_g}, & -\omega_0 - \omega_c < \omega < -\omega_0 + \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

Thus, since $G(\omega) = F(\omega)H(\omega)$,

$$G(\omega) = \begin{cases} \frac{1}{2}F_l(\omega - \omega_0)e^{\omega_0 t_p + (\omega - \omega_0)t_g}, & \omega_0 - \omega_c < \omega < \omega_0 + \omega_c \\ \frac{1}{2}F_l(\omega + \omega_0)e^{-\omega_0 t_p + (\omega + \omega_0)t_g}, & -\omega_0 - \omega_c < \omega < -\omega_0 + \omega_c \end{cases} \tag{2.20}$$

Calculating $g(t)$ using the inverse Fourier transform

$$\begin{aligned} g(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)H(\omega) \\ &= \frac{1}{2}\frac{1}{2\pi}[\int_{\omega_0 - \omega_c}^{\omega_0 + \omega_c} F_l(\omega - \omega_0)e^{j[(\omega - \omega_0)t_g + \omega_0 t_p]}e^{j\omega t}d\omega \\ &\quad + \int_{-\omega_0 - \omega_c}^{-\omega_0 + \omega_c} F_l(\omega + \omega_0)e^{j[(\omega + \omega_0)t_g - \omega_0 t_p]}e^{j\omega t}d\omega] \end{aligned} \tag{2.21}$$

Making the change in variables $u = \omega - \omega_0$ and $v = \omega + \omega_0$yields

$$\begin{aligned} g(t) &= \frac{1}{2}\frac{1}{2\pi}[\int_{-\omega_c}^{\omega_c} F_l(u)e^{j[ut_g + \omega_0 t_p]}e^{jut}e^{j\omega_0 t}du \\ &\quad + \int_{-\omega_c}^{\omega_c} F_l(v)e^{j[vt_g - \omega_0 t_p]}e^{jvt}e^{-j\omega_0 t}dv] \end{aligned} \tag{2.22}$$

Combining the integrals and performing some algebra gives

$$\begin{aligned} g(t) &= \frac{1}{2}\frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} F_l(\omega)e^{j\omega t_g}e^{j\omega t}[e^{j\omega_0 t_p}e^{j\omega_0 t}+e^{-j\omega_0 t_p}e^{-j\omega_0 t}]d\omega \\ &= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} F_l(\omega)\cos[\omega_0(t+t_p)]e^{j\omega(t+t_g)}d\omega \\ &= \cos[\omega_0(t+t_p)]\frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} F_l(\omega)e^{j\omega(t+t_g)}d\omega \\ &= \cos[\omega_0(t+t_p)]f_l(t+t_g) \qquad (2.23) \end{aligned}$$

which is the desired result.

The significance of the result in (2.23) is clear. The shape of the signal envelope due to $f_l(t)$ is unchanged and shifted in time by t_g. The carrier, however, is shifted in time by t_p (which in general is not equal to t_g). Consequently, the overall appearance of the ouput signal is changed with respect to that of the input due to the difference in phase shift between the carrier and the envelope. This phenomenon is illustrated in Figures **??**-**??**. Figure **??** illustrates `exec('group1_5.code')` Modulated Exponential Signal f0.1.1

a narrowband signal which consists of a sinusoid modulated by an envelope. The envelope is an decaying exponential and is displayed in the figure as the dotted curve.

Figure **??** shows the band pass filter used to filter the signal in Figure **??**. The filter magnitude is plotted as the solid curve and the filter phase is plotted as the dotted curve. `exec('group1_5.code')` Constant Phase Band Pass Filter f0.1.2

Notice that since the phase is a constant function that $t_g = 0$. The value of the phase delay is $t_p = \pi/2$. As is expected, the filtered output of the filter consists of the same signal as the input except that the sinusoidal carrier is now phase shifted by $\pi/2$. This output signal is displayed in Figure **??** as the solid curve. For reference the input signal is plotted as the dotted curve. `exec('group1_5.code')` Carrier Phase Shift by $t_p = \pi/2$ f0.1.3

To illustrate the effect of the group delay on the filtering process a new filter is constructed as is displayed in Figure **??**. `exec('group1_5.code')` Linear Phase Band Pass Filter f0.1.4

Here the phase is again displayed as the dotted curve. The group delay is the slope of the phase curve as it passes through zero in the pass band region of the filter. Here $t_g = -1$ and $t_p = 0$. The result of filtering with this phase curve is display in Figure **??**. As expected, the envelope is shifted but the sinusoid is not shifted within the reference frame of the window. The original input signal is again plotted as the dotted curve for reference. `exec('group1_5.code')` Envelope Phase Shift by $t_g = -1$ f0.1.5

The Function `group`

As can be seen from the explanation given in this section, it is preferable that the group delay of a filter be constant. A non-constant group delay tends to cause signal deformation. This is due to the fact that the different frequencies which compose the signal are time shifted by different amounts according to the value of the group delay at that frequency. Consequently, it is valuable to examine the group delay of filters during the design procedure. The function `group` accepts filter parameters in several formats as input and returns the group delay as output. The syntax of the function is as follows:

```
-->[tg,fr]=group(npts,h)
```

The group delay `tg` is evaluated in the interval [0,.5) at equally spaced samples contained in `fr`. The number of samples is governed by `npts`. Three formats can be used for the specification of the filter.

The filter h can be specified by a vector of real numbers, by a rational polynomial representing the z-transform of the filter, or by a matrix polynomial representing a cascade decomposition of the filter. The three cases are illustrated below.

The first example is for a linear-phase filter designed using the function wfir

as can be seen in Figure ?? exec('group6_8.code') Group Delay of Linear-Phase Filter f0.1.6

the group delay is a constant, as is to be expected for a linear phase filter. The second example specifies a rational polynomial for the filter transfer function:

The plot in Figure ?? gives the result of this calculation. exec('group6_8.code') Group Delay of Filter (Rational Polynomial) f0.1.7

Finally, the third example gives the transfer function of the filter in cascade form.

The result is shown in Figure ??. The cascade realization is known for numerical stability. exec('group6_8.code') Group Delay of Filter (Cascade Realization) f0.1.8

2.1.3 Appendix: Scilab Code Used to Generate Examples

The following listing of Scilab code was used to generate the examples of the this section.

```
//exec('group1_5.code')
//create carrier and narrow band signal
   xinit('group1.ps');
   wc=1/4;
   x=sin(2*%pi*(0:54)*wc);
   y=exp(-abs(-27:27)/5);
   f=x.*y;
   plot([1 1 55],[1 -1 -1]),
   nn=prod(size(f))
   plot2d((1:nn)',f',[2],"000"),
   nn=prod(size(y))
   plot2d((1:nn)',y',[3],"000"),
   plot2d((1:nn)',-y',[3],"000"),
   xend(),
   xinit('group2.ps');
//make band pass filter

   [h w]=wfir('bp',55,[maxi([wc-.15,0]),mini([wc+.15,.5])],'kr',60.);

//create new phase function with only phase delay

   hf=fft(h,-1);
   hm=abs(hf);
   hp=%pi*ones(1:28);//tg is zero
   hp(29:55)=-hp(28:-1:2);
   hr=hm.*cos(hp);
   hi=hm.*sin(hp);
   hn=hr+%i*hi;
   plot([1 1 55],[4 -4 -4]),
   plot2d([1 55]',[0 0]',[1],"000"),
   nn=prod(size(hp))
   plot2d((1:nn)',hp',[2],"000"),
   nn=prod(size(hm))
   plot2d((1:nn)',2.5*hm',[1],"000"),
   xend(),
   xinit('group3.ps');
//filter signal with band pass filter

   ff=fft(f,-1);
   gf=hn.*ff;
   g=fft(gf,1);
   plot([1 1 55],[1 -1 -1]),
   nn=prod(size(g))
   plot2d((1:nn)',real(g)',[2],"000"),
   nn=prod(size(f))
   plot2d((1:nn)',f',[1],"000"),

   xend(),

//create new phase function with only group delay
   xinit('group4.ps');
   tg=-1;
   hp=tg*(0:27)-tg*12.*ones(1:28)/abs(tg);//tp is zero
   hp(29:55)=-hp(28:-1:2);
   hr=hm.*cos(hp);
   hi=hm.*sin(hp);
   hn=hr+%i*hi;
   plot([1 1 55],[15 -15 -15]),
   plot2d([1 55]',[0 0]',[1],"000"),
   nn=prod(size(hp))
   plot2d((1:nn)',hp',[2],"000"),
   nn=prod(size(hm))
   plot2d((1:nn)',10*hm',[1],"000"),

   xend(),
   xinit('group5.ps');
//filter signal with band pass filter

   ff=fft(f,-1);
   gf=hn.*ff;
   g=fft(gf,1);
   plot([1 1 55],[1 -1 -1]),
   nn=prod(size(g))
```

```
plot2d((1:nn)',real(g)',[2],"000"),
nn=prod(size(f))
plot2d((1:nn)',f',[1],"000"),
xend(),
```

2.2 Sampling

The remainder of this section explains in detail the relationship between continuous and discrete signals.

To begin, it is useful to examine the Fourier transform pairs for continuous and discrete time signals. For $x(t)$ and $X(\Omega)$ a continuous time signal and its Fourier transform, respectively, we have that

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt \tag{2.24}$$

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)e^{j\Omega t}d\Omega. \tag{2.25}$$

For $x(n)$ and $X(\omega)$ a discrete time signal and its Fourier transform, respectively, we have that

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{2.26}$$

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}d\omega. \tag{2.27}$$

The discrete time signal, $x(n)$, is obtained by sampling the continuous time signal, $x(t)$, at regular intervals of length T called the sampling period. That is,

$$x(n) = x(t)|_{t=nT} \tag{2.28}$$

We now derive the relationship between the Fourier transforms of the continuous and discrete time signals. The discussion follows [21].

Using (2.28) in (2.25) we have that

$$x(n) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)e^{j\Omega nT}d\Omega. \tag{2.29}$$

Rewriting the integral in (2.29) as a sum of integrals over intervals of length $2\pi/T$ we have that

$$x(n) = \frac{1}{2\pi}\sum_{r=-\infty}^{\infty}\int_{(2\pi r-\pi)/T}^{(2\pi r+\pi)/T} X(\Omega)e^{j\Omega nT}d\Omega \tag{2.30}$$

or, by a change of variables

$$x(n) = \frac{1}{2\pi}\sum_{r=-\infty}^{\infty}\int_{-\pi/T}^{\pi/T} X(\Omega + \frac{2\pi r}{T})e^{j\Omega nT}e^{j2\pi nr}d\Omega. \tag{2.31}$$

Interchanging the sum and the integral in (2.31) and noting that $e^{j2\pi nr} = 1$ due to the fact that n and r are always integers yields

$$x(n) = \frac{1}{2\pi}\int_{-\pi/T}^{\pi/T}[\sum_{r=-\infty}^{\infty} X(\Omega + \frac{2\pi r}{T})]e^{j\Omega nT}d\Omega. \tag{2.32}$$

Finally, the change of variables $\omega = \Omega T$ gives

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi}[\frac{1}{T}\sum_{r=-\infty}^{\infty} X(\frac{\omega}{T} + \frac{2\pi r}{T})]e^{j\omega n}d\omega \tag{2.33}$$

which is identical in form to (2.27). Consequently, the following relationship exists between the Fourier transforms of the continuous and discrete time signals:

$$\begin{aligned} X(\omega) &= \frac{1}{T}\sum_{r=-\infty}^{\infty} X(\frac{\omega}{T} + \frac{2\pi r}{T}) \\ &= \frac{1}{T}\sum_{r=-\infty}^{\infty} X(\Omega + \frac{2\pi r}{T}). \end{aligned} \tag{2.34}$$

From (2.34) it can be seen that the Fourier transform of $x(n)$, $X(\omega)$, is periodic with period $2\pi/T$. The form of $X(\omega)$ consists of repetitively shifting and superimposing the Fourier transform of $x(t)$, $X(\Omega)$, scaled by the factor $1/T$. For example, if $X(\Omega)$ is as depicted in Figure **??**, where the highest non-zero frequency of $X(\Omega)$ is denoted by $\Omega_c = 2\pi f_c$, then there are two possibilities for $X(\omega)$. If $\pi/T > \Omega_c = 2\pi f_c$ then $X(\omega)$ is as in Figure **??**, and, if $\pi/T < \Omega_c = 2\pi f_c$, then $X(\omega)$ is as in Figure **??**. That is to say that if the sampling frequency $f_s = 1/T$ is greater than twice the highest frequency in $x(t)$ then there is no overlap in the shifted versions of $X(\Omega)$ in (2.34). However, if $f_s < 2f_c$ then the resulting $X(\omega)$ is composed of overlapping versions of $X(\Omega)$.

`exec('sample1.code')` Frequency Response $X(\Omega)$ f00.1

`exec('sample2.code')` Frequency Response $x(\omega)$ With No Aliasing f0.2

`exec('sample3.code')` Frequency Response $x(\omega)$ With Aliasing f0.3

to say that if the sampling frequency $f_s = 1/T$ is greater than twice the highest frequency in $x(t)$ then there is no overlap in the shifted versions of $X(\Omega)$ in (2.34). However, if $f_s < 2f_c$ then the resulting $X(\omega)$ is composed of overlapping versions of $X(\Omega)$.

The sampling rate $T = 1/(2f_c)$ is the well known Nyquist sampling rate and any signal sampled at a rate higher than the Nyquist rate retains all of the information that was contained in the original unsampled signal. It can be concluded that sampling can retain or alter the character of the original continuous time signal. If sampling is performed at more than twice the highest frequency in $x(t)$ then the signal nature is retained. Indeed, the original signal can be recuperated from the sampled signal by low pass filtering (as is demonstrated below). However, if the signal is undersampled this results in a signal distortion known as aliasing.

To recuperate the original analog signal from the sampled signal it is assumed that $\Omega_c < \pi/T$ (i.e., that the signal is sampled at more than twice its highest frequency). Then from (2.34)

$$X(\Omega) = TX(\omega) \tag{2.35}$$

in the interval $-\pi/T \leq \Omega \leq \pi/T$. Plugging (2.35) into (2.25) yields

$$x(t) = \frac{1}{2\pi}\int_{-\pi/T}^{\pi/T} TX(\omega)e^{j\Omega t}d\Omega. \tag{2.36}$$

Replacing $X(\omega)$ by (2.26) and using (2.28) we have that

$$x(t) = \frac{T}{2\pi}\int_{-\pi/T}^{\pi/T}[\sum_{-\infty}^{\infty} x(nT)e^{-j\Omega nT}]e^{j\Omega t}d\Omega. \tag{2.37}$$

Interchanging the sum and the integral gives

$$x(t) = \sum_{-\infty}^{\infty} x(nT)[\frac{T}{2\pi}\int_{-\pi/T}^{\pi/T} e^{j\Omega(t-nT)}d\Omega]. \tag{2.38}$$

The expression in brackets in (2.38) can be recognized as a time shifted inverse Fourier transform of a low pass filter with cut-off frequency π/T. Consequently, (2.38) is a convolution between the sampled signal and a low pass filter, as was stated above.

We now illustrate the effects of aliasing. Since square integrable functions can always be decomposed as a sum of sinusoids the discussion is limited to a signal which is a cosine function. The results of what happens to a cosine signal when it is undersampled is directly extensible to more complicated signals.

We begin with a cosine signal as is illustrated in Figure **??**. `exec('sample4.code')` Cosine Signal f0.4

The cosine in Figure **??** is actually a sampled signal which consists of 5000 samples. One period of the cosine in the figure is 200 samples long, consequently, the Nyquist sampling rate requires that we retain one sample in every 100 to retain the character of the signal. By sampling the signal at a rate less than the Nyquist rate it would be expected that aliasing would occur. That is, it would be expected that the sum of two cosines would be evident in the resampled data. Figure **??** illustrates the data resulting from sampling the cosine in Figure **??** at a rate of ones every 105 samples. `exec('sample5.code')` Aliased Cosine Signal f0.5

As can be seen in Figure **??**, the signal is now the sum of two cosines which is illustrated by the beat signal illustrated by the dotted curves.

2.3 Decimation and Interpolation

2.3.1 Introduction

There often arises a need to change the sampling rate of a digital signal. The Fourier transform of a continuous-time signal, $x(t)$, and the Fourier transform of the discrete-time signal, $x(nT)$, obtained by sampling $x(t)$ with frequency $1/T$. are defined, respectively, in (2.39) and (2.40) below

$$\hat{X}(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{2.39}$$

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x(nT)e^{-j\omega T}. \tag{2.40}$$

The relationship between these two transforms is (see [21]) :

$$X(e^{j\omega T}) = \frac{1}{T}\sum_{r=-\infty}^{\infty}\hat{X}(\frac{j\omega}{T}+\frac{j2\pi r}{T}). \tag{2.41}$$

Figure **??** illustrates the magnitude of the Fourier transform $\hat{X}(\omega)$ of a signal $x(t)$. Figure **??** shows two periods of the associated Fourier transform $X(e^{jwT})$ of $x(nT)$ where the sampling frequency was taken to be the Nyquist rate. As indicated by (2.41), the magnitude of $X(e^{jwT})$ with respect to the magnitude of $\hat{X}(\omega)$ is scaled by $1/T$. `exec('intdec1_4.code')` Fourier Transform of a Continuous Time Signal f0.1

`exec('intdec1_4.code')` Fourier Transform of the Discrete Time Signal f1.1

Furthermore, $X(e^{jwT})$ is periodic with period $2\pi/T$. If we take $1/T \geq \pi/\Omega$, where Ω is the highest non-zero frequency of $X(\omega)$, then no aliasing occurs in sampling the continuous-time signal. When this is the case one can, in principle, perfectly reconstruct the original continuous-time signal $x(t)$ from its samples $x(nT)$ using

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT) \frac{\sin[(\pi/T)(t-nT)]}{(\pi/T)(t-nT)}. \tag{2.42}$$

Consequently, one could obtain $x(t)$ sampled at a different sampling rate T' from the sampled signal $x(nT)$ by using (2.42) to reconstruct $x(t)$ and then resampling. In practice, however, this is impractical. It is much more convenient to keep all operations in the digital domain once one already has a discrete-time signal.

The Scilab function `intdec` accomplishes a sampling rate change by interpolation and decimation. The interpolation takes the input signal and produces an output signal which is sampled at a rate L (an integer) times more frequently than the input. Then decimation takes the input signal and produces an output signal which is sampled at a rate M (also an integer) times less frequently than the input.

2.3.2 Interpolation

In interpolating the input signal by the integer L we wish to obtain a new signal $x(nT')$ where $x(nT')$ would be the signal obtained if we had originally sampled the continuous-time signal $x(t)$ at the rate $1/T' = L/T$. If the original signal is bandlimited and the sampling rate $f = 1/T$ is greater than twice the highest frequency of $x(t)$ then it can be expected that the new sampled signal $x(nT')$ (sampled at a rate of $f' = 1/T' = L/T = Lf$) could be obtained directly from the discrete signal $x(nT)$.

An interpolation of $x(nT)$ to $x(nT')$ where $T' = T/L$ can be found by inserting $L-1$ zeros between each element of the sequence $x(nT)$ and then low pass filtering. To see this we construct the new sequence $v(nT')$ by putting $L-1$ zeros between the elements of $x(nT)$

$$v(nT') = \begin{cases} x(nT/L), & n = 0, \pm L, \pm 2L, \ldots \\ 0, & \text{otherwise.} \end{cases} \tag{2.43}$$

Since $T' = T/L$, $v(nT')$ is sampled L times more frequently than $x(nT)$. The Fourier transform of (2.43) yields

$$\begin{aligned} V(e^{j\omega T'}) &= \sum_{n=-\infty}^{\infty} v(nT') e^{-j\omega nT'} \\ &= \sum_{n=-\infty}^{\infty} x(nT) e^{-j\omega nLT'} \\ &= \sum_{n=-\infty}^{\infty} x(nT) e^{-j\omega nT} \\ &= X(e^{j\omega T}). \end{aligned} \tag{2.44}$$

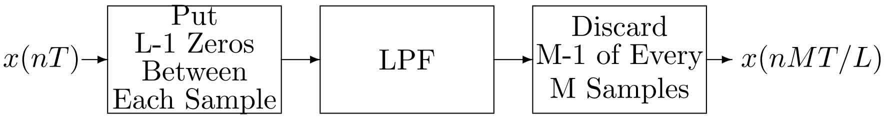
$x(nT)$
Put L-1 Zeros Between Each Sample
LPF
Discard M-1 of Every M Samples
$x(nMT/L)$

Figure 2.1: Block Diagram of Interpolation and Decimation

From (2.44) it can be seen that $V(e^{j\omega T'})$ is periodic with period $2\pi/T$ and, also, period $2\pi/T' = 2\pi L/T$. This fact is illustrated in Figure **??** where $L = 3$. Since the sampling frequency of V is $1/T'$ we see that by filtering $v(nT')$ with a low `exec('intdec1_4.code')` Fourier Transform of $v(nT')$ f2.1

pass filter with cut-off frequency at π/T we obtain exactly the interpolated sequence, $x(nT')$, which we seek (see Figure **??**), except for a scale factor of L (see (2.41)). `exec('intdec1_4.code')` Fourier Transform of $x(nT')$ f3.1

2.3.3 Decimation

Where the object of interpolation is to obtain $x(nT')$ from $x(nT)$ where $T' = L/T$, the object of decimation is to find $x(nT'')$ from $x(nT)$ where $T'' = MT$, M an integer. That is, $x(nT'')$ should be equivalent to a sequence obtained by sampling $x(t)$ M times less frequently than that for $x(nT)$. Obviously this can be accomplished by keeping only every M^{th} sample of $x(nT)$. However, if the sampling frequency $1/T$ is close to the Nyquist rate then keeping only every M^{th} sample results in aliasing. Consequently, low pass filtering the sequence $x(nT)$ before discarding $M-1$ of each M points is advisable. Assuming that the signal $x(nT)$ is sampled at the Nyquist rate, the cut-off frequency of the low pass filter must be at $\pi/(MT)$.

2.3.4 Interpolation and Decimation

To change the sampling rate of a signal by a non-integer quantity it suffices to perform a combination of the interpolation and decimation operations. Since both operations use a low-pass filter they can be combined as illustrated in the block diagram of Figure 2.1. The Scilab function `intdec` begins by designing a low-pass filter for the diagram illustrated in the figure. It accomplishes this by using the `wfir` filter design function. This is followed by taking the Fourier transform of both the input signal and the low-pass filter (whose magnitude is first scaled by L) by using the `fft` function. Care must be taken to obtain a linear convolution between the two sequences by adding on an appropriate number of zeros to each sequence before the FFT is performed. After multiplying the two transformed sequences together an inverse Fourier transform is performed. Finally, the output is obtained by discarding $M-1$ of each M points. The cut-off frequency of the low pass filter is π/T if $L > M$ and is $(L\pi)/(MT)$ if $L < M$.

The practical implementation of the interpolation and decimation procedure is as follows. If the length of the input is N then after putting $L-1$ zeros between each element of x the resulting sequence will be of length $(N-1)L+1$. This new sequence is then lengthened by $K-1$ zeros where K is the length of the low pass filter. This lengthening is to obtain a linear convolution between the input and the low pass filter with the use of the FFT. The cut-off frequency of the low pass filter is chosen to be $(.5N)/[(N-1)L+K]$ if $L > M$ and $(.5NL)/(M[(N-1)L+K])$ if $L < M$. The FFT's of the two modified sequences are multiplied element by element and then are inverse Fourier transformed. The resulting sequence is of length $(N-1)L+K$. To obtain a sequence of length of $(N-1)L+1$ elements, $(K-1)/2$ elements are discarded off of each end. Finally, $M-1$ out of every M elements are discarded to form the output sequence.

2.3.5 Examples using `intdec`

Here we take a 50-point sequence assumed to be sampled at a 10kHz rate and change it to a sequence sampled at 16kHz. Under these conditions we take $L = 8$ and $M = 5$. The sequence, $x(nT)$, is illustrated in Figure ??. The discrete Fourier transform of x(nT) is shown in `exec('intdec5_10.code')` The Sequence $x(nT)$ f55.1

`exec('intdec5_10.code')` The DFT of $x(nT)$ f6.1

Figure ??. As can be seen, $x(nT)$ is a bandlimited sequence. A new sequence $v(nT')$ is created by putting 7 zeros between each element of $x(nT)$. We use a Hamming windowed lowpass filter of length 33 (Figure ??) `exec('intdec5_10.code')` Low Pass Filter f7.1

to filter $v(nT')$. The discrete Fourier transform of $v(nT')$ is illustrated in Figure ??. As is to be expected, the Fourier transform of $v(nT')$ looks like the Fourier transform of $x(nT)$ repeated 8 times. `exec('intdec5_10.code')` DFT of $v(nT')$ f8.1

The result of multiplying the magnitude response of the filter with that of the sequence $v(nT')$ is shown in Figure ??. Since the low pass filter is not ideal the resulting filtered sequence has some additional high frequency energy in it (i.e., the small lobes seen in Figure ??). `exec('intdec5_10.code')` Filtered Version of V f9.1

Finally, after taking the inverse discrete Fourier transform and discarding 4 out of every 5 samples we obtain the sequence illustrated in Figure ??. `exec('intdec5_10.code')` Sequence $x(nMT/L)$ f10.1

2.4 The DFT and the FFT

2.4.1 Introduction

The FFT ("Fast Fourier Transform") is a computationally efficient algorithm for calculating the DFT ("Discrete Fourier Transform") of finite length discrete time sequences. Calculation of the DFT from its definition requires order N^2 multiplications whereas the FFT requires order $N\log_2 N$ multiplications. In this section we discuss several uses of the DFT and some examples of its calculation using the FFT primitive in Scilab.

We begin with the definition of the DFT for a finite length sequence, $x(n)$, of length N,

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk}. \tag{2.45}$$

A careful examination of (2.45) reveals that $X(k)$, the DFT of $x(n)$, is periodic with period N (due to the fact that for fixed n the term $\exp(-j2\pi nk/N)$ is periodic with period N). That $X(k)$ is periodic also follows from the fact that (2.45) can be interpreted as samples of the z-transform of $x(n)$ at N equally spaced spaced samples on the unit circle. For reasons of symmetry, the DFT is defined to consist of the N distinct points of $X(k)$ for $k = 0, 1, \ldots, N-1$.

The N points of the sequence $x(n)$ can be recovered from N points of $X(k)$. This recovery is called the inverse DFT and takes the form

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}nk}. \tag{2.46}$$

It should be noted that (2.46) yields a periodic sequence in n of period N and that it is the N values of $x(n)$, $n = 0, 1, \ldots, N-1$ which yield the desired finite length sequence.

In order to better understand the DFT, one can think of (2.45) as being represented by a matrix computation

$$\begin{bmatrix} X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j\frac{2\pi}{N}} & e^{-j\frac{4\pi}{N}} & \cdots & e^{-j\frac{2(N-1)\pi}{N}} \\ 1 & e^{-j\frac{4\pi}{N}} & e^{-j\frac{8\pi}{N}} & \cdots & e^{-j\frac{4(N-1)\pi}{N}} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{-j\frac{2(N-1)\pi}{N}} & e^{-j\frac{4(N-1)\pi}{N}} & \cdots & e^{-j\frac{(N-1)^2\pi}{N}} \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}. \tag{2.47}$$

The inverse DFT can be calculated in a similar fashion, where the matrix used is the Hermitian transpose of that in (2.47) times a factor of $1/N$. From (2.47) it can be seen that the DFT requires order N^2 multiplications. The most direct application of the DFT is the calculation of the spectrum of finite length discrete signals. In fact, the DFT is a projection of $x(n)$ onto the orthogonal basis consisting of the N complex exponentials $\exp(-j2\pi nk/N)$ indexed by k. Another important property of the DFT has to do with the inverse DFT of the product of two transformed sequences. Taking $x(n)$ and $h(n)$ to be two sequences of length N the DFT's of these two sequences are

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk} \tag{2.48}$$

and

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j\frac{2\pi}{N}nk}. \tag{2.49}$$

Taking the inverse DFT of the product $Y(k) = H(k)X(k)$ yields

$$\begin{aligned} y(n) &= \frac{1}{N} \sum_{k=0}^{N-1} H(k)X(k) e^{j\frac{2\pi}{N}nk} \\ &= \sum_{m=0}^{N-1} h(m) \sum_{r=0}^{N-1} x(r) \frac{1}{N} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}(n-m-r)k} \\ &= \sum_{m=0}^{N-1} h(m)x(n-m) \end{aligned} \tag{2.50}$$

where the last equality follows from the fact that

$$\frac{1}{N} \sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}(n-m-r)k} = \begin{cases} 1, & r = n-m \\ 0, & \text{otherwise} \end{cases}. \tag{2.51}$$

In (2.50) it should be noted that when the argument of x is outside of the the range $[0, N-1]$ that the value of x is obtained from evaluating the argument modulo N (which follows from the periodicity of the x obtained by using the inverse DFT). A very important application of the DFT is for calculating the interaction of discrete signals and discrete linear systems. When a discrete signal is introduced to the input of a linear system, the resulting output is a convolution of the input signal with the impulse response of the linear system. The convolution operation, which is discussed in more detail in the section on convolution, requires order N^2 multiplications where the signal and system impulse response are both of length N. Calculating the convolution by FFT requires order $N\log_2 N$ multiplications.

It is equally possible to compute a multi-dimensional DFT. For a multi-dimensional sequence $x(n_1, n_2, \ldots, n_M)$ the multi-dimensional DFT is defined by

$$\begin{aligned} X(k_1, k_2, \ldots, k_M) &= \\ \sum_{n_1=1}^{N_1-1} e^{-j\frac{2\pi}{N_1}n_1k_1} \sum_{n_2=1}^{N_2-1} e^{-j\frac{2\pi}{N_2}n_2k_2} &\cdots \sum_{n_M=1}^{N_M-1} e^{-j\frac{2\pi}{N_M}n_Mk_M} x(n_1, n_2, \ldots, n_M). \end{aligned} \tag{2.52}$$

The inverse multi-dimensional DFT is analogous to the calculation above with a change of sign for the complex exponentials and a factor of $1/(N_1N_2\cdots N_M)$.

The FFT algorithm is a computationally efficient way of calculating the DFT. The computational savings realized by the FFT are obtained by exploiting certain symmetries and periodicities which exist in the calculation of the DFT. To show how the FFT exploits these properties of the DFT we assume that $N = 2^\gamma$ for γ a positive integer and we calculate the DFT of $x(n)$ as follows

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk} \\ &= \sum_{n=even} x(n)e^{-j\frac{2\pi}{N}nk} + \sum_{n=odd} x(n)e^{-j\frac{2\pi}{N}nk} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x(2r)e^{-j\frac{2\pi}{N/2}rk} + e^{j\frac{2\pi}{N}k} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)e^{-j\frac{2\pi}{N/2}rk} \end{aligned} \tag{2.53}$$

where the final equality is obtained by making the change of variables $n = 2r$. The expression in (2.53) is composed of a sum of two $N/2$ point DFT's, one for the $N/2$ point sequence $x(0)$, $x(2)$, …, $x(N-2)$, and the other for the $N/2$ point sequence $x(1)$, $x(3)$, …, $x(N-1)$. An addition to the two $N/2$ point DFT's in (2.53), the calculation also requires N additional multiplications for the N terms $\exp(j2\pi k/N)$, $k = 0, 1, \ldots, N-1$.

The purpose of calculating the DFT as in (2.53) is that a computational savings has been realized. As has already been shown the calculation of the N point DFT from (2.45) requires order N^2 multiplications. Since (2.53) requires the calculation of two $N/2$ point DFT's plus N additional multiplications, the computational load is of order $2(N/2)^2 + N = N^2/2 + N$. For $\gamma > 1$ (i.e., for $N \geq 4$) we have realized a computational savings by using (2.53). Furthermore, the operation in (2.53) can be repeated in that each of the $N/2$ point DFT's can be split into two $N/4$ point DFT's plus N additional multiplications. This yields a computational complexity of $4(N/4)^2 + 2N$ multiplications. Continuing in this way $\gamma = \log_2 N$ times, the final result is an algorithm with computational complexity of $N\log_2 N$ multiplications.

The above discussion of the computational advantages of the FFT is based on the assumption that $N = 2^\gamma$. Similar developments of the FFT can be derived based on any prime factorization of N. The more prime factors N has the greater computational efficiency can be obtained in using the FFT. In fact, it may be useful in some applications to artificially extend the length of a sequence (by adding on zeros) in order that the length of the sequence will be more factorable. The FFT primitive in Scilab automatically accounts for the prime factorization of the sequence length.

2.4.2 Examples Using the `fft` Primitive

Two examples are presented in this section. The first example illustrates how to use the `fft` primitive to calculate a one-dimensional DFT. The second example calculates a three-dimensional DFT.

For the first example, data from a cosine function is passed to the `fft` primitive.

The cosine data is displayed in Figure **??**. resulting output from the `fft` primitive is displayed in Figure **??**. Figure **??** displays the magnitude of the DFT. `exec('fft1.code')` Cosine Signalf.fft1 `exec('fft2.code')` DFT of Cosine Signalf.fft2

Note, however, that since the cosine function is an even symmetric function, the DFT of the cosine is strictly real and, thus, the magnitude and the real part of the DFT are the same. Furthermore, since we are calculating a 64-point DFT of a cosine with frequency $2\pi/16$ it is expected that the DFT should have peaks at $k = 4$ and $k = 60$. This follows from the fact that the value $k = 64$ of the DFT corresponds to a frequency of 2π and, consequently, the value $k = 4$ must correspond to the frequency $2\pi/16$, which is the frequency of the signal under examination.

The second example calculates the DFT of a three-dimensional signal. The calculation proceeds as follows.

In the above series of calculations the signal y is three-dimensional and is represented by the two matrices `y1` and `y2`. The first dimension of y are the rows of the matrices, the second dimension of y are the columns of the matrices, and the third dimension of y are the sequence of matrices represented by `y1` and `y2`. The signal y is represented by the vector `y` which is in vector form. The DFT of y is calculated using the function `mfft` where `flag` $= -1$ and `dim` $= [2\ 3\ 2]$. Naturally, the DFT of the three-dimensional signal is itself three-dimensional. The result of the DFT calculation is represented by the two matrices `yf1` and `yf2`.

2.5 Convolution

2.5.1 Introduction

Given two continuous time functions $x(t)$ and $h(t)$, a new continuous time function can be obtained by convolving $x(t)$ with $h(t)$.

$$y(t) = \int_{-\infty}^{\infty} h(t-u)x(u)du. \tag{2.54}$$

An analogous convolution operation can be defined for discrete time functions. Letting $x(n)$ and $h(n)$ represent discrete time functions, by the discrete time convolution gives $y(n)$ is :

$$y(n) = \sum_{k=-\infty}^{\infty} h(n-k)x(k). \tag{2.55}$$

If $h(t)$ represents the impulse response of a linear, time-invariant system and if $x(t)$ is an input to this system then the output of the system, $y(t)$, can be calculated as the convolution between $x(t)$ and $h(t)$ (i.e., as in (2.54). Figure 2.2 illustrates how convolution is related to linear systems. If the system, $h(t)$ is causal (i.e., $h(t) = 0$ for $t < 0$) and, in addition, the signal $x(t)$ is applied to the system at time $t = 0$, then (2.54) becomes

$$y(t) = \int_{0}^{t} h(t-u)x(u)du. \tag{2.56}$$

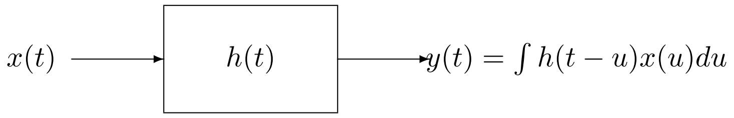

Figure 2.2: Convolution Performed by Linear System

Similarly, for $h(n)$ a time invariant, causal, discrete linear system with input $x(n)$ starting at time $n = 0$, the output $y(n)$ is the convolution

$$y(n) = \sum_{k=0}^{n} h(n-k)x(k). \tag{2.57}$$

An important property of the convolution operation is that the calculation can be effected by using Fourier transforms. This is due to the fact that convolution in the time domain is equivalent to multiplication in the frequency domain. Let $X(\omega)$, $H(\omega)$, and $Y(\omega)$ represent the Fourier transforms of $x(t)$, $h(t)$, and $y(t)$, respectively, where

$$Y(\omega) = \int_{-\infty}^{\infty} y(t)e^{-j\omega t}dt. \tag{2.58}$$

If the relationship in (2.54) is valid, then it also follows that

$$Y(\omega) = H(\omega)X(\omega). \tag{2.59}$$

There is an analogous relationship between the Fourier transform and convolution for discrete time signals. Letting $X(e^{j\omega})$, $H(e^{j\omega})$, and $Y(e^{j\omega})$ be the Fourier transforms of $x(n)$, $h(n)$, and $y(n)$, respectively, where, for example

$$Y(e^{j\omega}) = \sum_{n=-\infty}^{\infty} y(n)e^{-j\omega n} \tag{2.60}$$

we have that the discrete time convolution operation can be represented in the Fourier domain as

$$Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega}). \tag{2.61}$$

The fact that convolution in the time domain is equivalent to multiplication in the frequency domain means that the calculations in (2.54) and (2.55) can be calculated (at least approximately) by a computationally efficient algorithm, the FFT. This is accomplished by calculating the inverse DFT of the product of the DFT's of $x(n)$ and $h(n)$. Care must be taken to ensure that the resulting calculation is a linear convolution (see the section on the DFT and the FFT). The linear convolution is accomplished by adding enough zeros onto the two sequences so that the circular convolution accomplished by the DFT is equivalent to a linear convolution.

The convolution of two finite length sequences can be calculated by the Scilab function `convol`.

2.5.2 Use of the `convol` function

The `convol` function can be used following two formats. The first format calculates a convolution based on two discrete length sequences which are passed in their entirety to the function. The second format performs updated convolutions using the overlap-add method described in [21]. It is used when one of the sequences is very long and, consequently, cannot be passed directly to the function.

The syntax used for the function under the first format is

```
-->y=convol(h,x)
```

where both `h` and `x` are finite length vectors and `y` is a vector representing the resulting convolution of the inputs. An example of the use of the function under the first format is as follows.

The syntax used for the function under the second format is

```
-->[y,y1]=convol(h,x,y0)
```

where `y0` and `y1` are required to update the calculations of the convolution at each iteration and where the use of the second format requires the following function supplied by the user.

```
//exec('convol1.code')
x1=getx(xlen_1,xstart_1);
[y,y1]=convol(h,x1);
for k=2:nsecs-1,
   xk=getx(xlen_k,xstart_k);
   [y,y1]=convol(h,xk,y1);
end,
xn=getx(xlen_n,xstart_n);
y=convol(h,xn,y1);
```

where, `nsecs` is the number of sections of x to be used in the convolution calculation and, in addition, the user must supply a function `getx` which obtains segments of the data x following the format.

```
function [xv]=getx(xlen,xstart)
   .
   .
   .
```

where `xlen` is the length of data requested and `xstart` is the length of the data vector to be used.

2.6 The Chirp Z-Transform

2.6.1 Introduction

The discrete Fourier transform (DFT) of a finite length, discrete time signal, $x(n)$, is defined by

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)e^{-j(2\pi nk)/N} \\ k &= 0,1,\ldots,N-1 \end{aligned} \tag{2.62}$$

and the z-transform of $x(n)$ is given by

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x(n)z^{-n} \\ &= \sum_{n=0}^{N-1} x(n)z^{-n} \end{aligned} \qquad (2.63)$$

The $N-1$ points of the DFT of $x(n)$ are related to the z-transform of $x(n)$ in that they are samples of the z-transform taken at equally spaced intervals on the unit circle in the z-plane.

There are applications [25] where it is desired to calculate samples of the z-transform at locations either off the unit circle or at unequally spaced angles on the unit circle. The chirp z-transform (CZT) is an efficient algorithm which can be used for calculating samples of some of these z-transforms. In particular, the CZT can be used to efficiently calculate the values of the z-transform of a finite-length, discrete-time sequence if the z-transform points are of the form

$$z_k = AW^{-k} \qquad (2.64)$$

where

$$\begin{aligned} A &= A_0 e^{j\theta} \\ W &= W_0 e^{-j\phi} \end{aligned} \qquad (2.65)$$

and where A_0 and W_0 are real valued constants and θ and ϕ are angles. `exec('czt1.code')` Samples of the z-transform on Spiralsf4.1 The set of points $\{z_k\}$ lie on a spiral where z_0 is at distance A_0 from the origin and at angle θ from the x-axis. The remaining points are located at equally spaced angles, ϕ, and approach the origin for $W_0 > 1$, move away from the origin for $W_0 < 1$, and remain on a circle of radius A_0 for $W_0 = 1$. Figure **??** shows the location of samples of the z-transform for $W_0 < 1$ on the left hand side of the figure and of $W_0 < 1$ on the right hand side of the figure. In both parts of the figure the position of z_0 is indicated by the sample connected to the origin by a straight line.

2.6.2 Calculating the CZT

Calculating samples of the z-transform of $x(n)$ at the M points designated in (2.65) requires that

$$X(z_k) = \sum_{n=0}^{N-1} x(n)A^{-n}W^{nk}, \quad k = 0, 1, \ldots, M-1 \qquad (2.66)$$

where N is the length of $x(n)$. Using the identity

$$nk = \frac{1}{2}[n^2 + k^2 - (k-n)^2] \qquad (2.67)$$

in (2.66) yields

$$\begin{aligned} X(z_k) &= \sum_{n=0}^{N-1} x(n)A^{-n}W^{\frac{1}{2}n^2}W^{\frac{1}{2}k^2}W^{-\frac{1}{2}(k-n)^2} \\ &= W^{\frac{1}{2}k^2}\sum_{n=0}^{N-1}[x(n)A^{-n}W^{\frac{1}{2}n^2}W^{-\frac{1}{2}(k-n)^2}]. \end{aligned} \qquad (2.68)$$

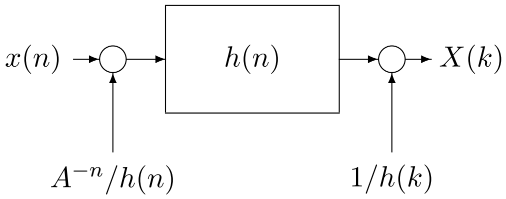

Figure 2.3: Filter Realization of CZT

It can be seen that imbedded in (2.68) is a convolution of two sequences $g(n)$ and $h(n)$ where

$$g(n) = x(n)A^{-n}W^{\frac{1}{2}n^2} \tag{2.69}$$

and

$$h(n) = W^{-\frac{1}{2}n^2}. \tag{2.70}$$

Consequently, (2.68) can be represented by the block diagram in Figure 2.3. (The circular junctions in Figure 2.3 represent multiplication of the two incoming signals).

The convolution in (2.68) can be efficiently implemented using an FFT. Since the input sequence is of length N and the output sequence is of length M, it is necessary to use $N+M-1$ elements from $h(n)$. These $N+M-1$ elements are $h(-N+1), h(-N+2), \ldots, h(n), \ldots, h(M-2), h(M-1)$. After taking the product of the $N+M-1$ point FFT of $h(n)$ and of $g(n)$ (where $M-1$ zero points have been added to the end of $g(n)$), the inverse FFT yields a circular convolution of $h(n)$ and $g(n)$. Care must be taken to choose the correct M points corresponding to the linear convolution desired. The function `czt` implements the chirp z-transform.

2.6.3 Examples

The first example presented here calculates the CZT of the sequence $x(n) = n$ for $n = 0, 1, \ldots, 9$ where ten points are calculated in the z-plane and the parameter values are $W0 = 1$, $\phi = 2\pi/10$, $A0 = 1$, and $\theta = 0$. This example should yield results identical to those obtained by taking the FFT of the sequence $x(n)$. The sequence of commands is as follows,

As can be verified using the function `fft`, the above result is identical to that obtained by taking the FFT of the sequence $x(n)$ which is shown below, The second example calculates the DFT of the same sequence, $x(n)$, above, however, just at five equally spaced points in $[-\pi/4, \pi/4]$ on the unit circle in the z-plane. The spacing between the points is $\pi/8$ for five points in $[-\pi/4, \pi/4]$. The result is

Now taking a sixteen point FFT of the sequence $x(n)$ (accomplished by adding six zeros to the end of the sequence $x(n)$) it can be seen that the CZT computed above yields exactly the desired points on the unit circle in the z-plane. That is to say that the last three points of `czx` correspond to the first three points of the FFT of $x(n)$ and the first two points of `czx` correspond to the last two points of the FFT.

Chapter 3

Design of Finite Impulse Response Filters

3.1 Windowing Techniques

In theory, the design of FIR filters is straightforward. One takes the inverse Fourier transform of the desired frequency response and obtains the discrete time impulse response of the filter according to (3.1)

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega \quad -\infty < n < \infty \tag{3.1}$$

The problem, in practice, is that for many filters of interest the resulting impulse response is infinite and non-causal. An example of this is the low pass filter which, given its cut-off frequency, ω_c, is defined by

$$H(\omega|\omega_c) = \left\{ \begin{array}{ll} 1, & |\omega| \leq \omega_c \\ 0, & \text{otherwise} \end{array} \right. \tag{3.2}$$

The associated impulse response is obtained by applying (3.1) to (3.2) which yields

$$h(n|\omega_c) = \frac{1}{\pi n}\sin(\omega_c n) \quad -\infty < n < \infty \tag{3.3}$$

A technique for obtaining a finite length implementation to (3.3) is to take the N elements of $h(n)$ which are centered around $n = 0$ and to discard all the remaining elements. This operation can be represented by multiplying the sequence in (3.3) by an appropriately shifted version of a rectangular window of the form

$$R_N(n) = \left\{ \begin{array}{ll} 1, & 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{array} \right. \tag{3.4}$$

The magnitude of the resulting windowed sequence frequency response is depicted in Figure **??** superimposed on the ideal frequency response (the dotted curve). The filter illustrated in Figure **??** has length $N = 33$ and a cut-off frequency of $\omega_c = .2$. As can be seen, the `exec('fir1.code')` Rectangularly windowed low-pass filterf1 approximation is marked by a ripple in both the pass and stop bands. This ripple finds its greatest deviations near the discontinuity at ω_c. The observed ripple is due to the convolution of the ideal frequency response given by (3.2) with the frequency response of the rectangular window. For many applications the ripples in the frequency response of Figure **??** are unacceptable.

It is possible to decrease the amount of rippling by using different types of windows. The performance of a window is governed by its frequency response. Since the frequency response of the window is convolved with the desired frequency response the objective is to find a window which has a frequency response which is as impulsive as possible. That is, the frequency response should have a narrow main lobe with most of the energy in this lobe and side lobes which are as small as possible. The width of the main lobe governs, for example, the width of the transition band between the pass and stop bands of a low pass filter. The side lobes govern the amount ripple in the pass and stop bands. The area under the main lobe governs the amount of rejection available in the stop bands.

The choice of window in the design process is a matter of trading off, on one hand, the effects of transition band width, ripple, and rejection in the stop band with, on the other hand, the filter length and the window type.

3.1.1 Filter Types

The Scilab function `wfir` designs four different types of FIR linear phase filters: low pass, high pass, band pass, and stop band filters. The impulse response of the three latter filters can be obtained from the low pass filter by simple relations and the impulse response of the low pass filter is given in (3.3).

To show the relationship between the four filter types we first examine Figure **??** which illustrates the frequency response of a low pass filter with cut off frequency denoted by ω_l. `exec('fir2_5.code')` Frequency response of a low pass filter f2 The frequency response of a high pass filter is illustrated in Figure **??** `exec('fir2_5.code')` Frequency response of a high pass filter f3 where ω_h denotes, also, the cut off frequency. Taking the functional form of the low pass filter to be $H(\omega|\omega_l)$ and that of the high pass filter to be $G(\omega|\omega_h)$, the relationship between these two frequency responses is

$$G(\omega|\omega_h) = 1 - H(\omega|\omega_h). \tag{3.5}$$

Using the result in (3.3), the impulse response of the high pass filter is given by

$$\begin{aligned} g(n|\omega_h) &= \delta(n) - h(n|\omega_h) & (3.6)\\ &= \delta(n) - \frac{1}{n\pi}\sin(\omega_h n) & (3.7) \end{aligned}$$

where $\delta(n) = 1$ when $n = 0$ and is zero otherwise.

For a band pass filter, as illustrated in Figure **??**, `exec('fir2_5.code')` Frequency response of a band pass filter f4 the functional form of the frequency response, $F(\omega|\omega_l, \omega_h)$ can be obtained by shifting the low pass filter two times as follows

$$\begin{aligned} F(\omega|\omega_l,\omega_h) &= H(\omega-\omega_1|\omega_2) + H(\omega+\omega_1|\omega_2) & (3.8)\\ \omega_1 &= \frac{1}{2}(\omega_l+\omega_h) & (3.9)\\ \omega_2 &= \frac{1}{2}(\omega_l-\omega_h). & (3.10) \end{aligned}$$

Thus, the impulse response of the band pass filter is

$$\begin{aligned} f(n|\omega_l,\omega_h) &= e^{j\omega_1 n}h(n|\omega_2) + e^{-j\omega_1 n}h(n|\omega_2) & (3.11)\\ &= \frac{2}{n\pi}\cos(\omega_1 n)\sin(\omega_2 n). & (3.12) \end{aligned}$$

Finally, the stop band filter illustrated in Figure **??** `exec('fir2_5.code')` Frequency response of a stop band filter f5 can be obtained from the band pass filter by the relation

$$D(\omega|\omega_l,\omega_h) = 1 - F(\omega|\omega_l,\omega_h) \tag{3.13}$$

where $D(\omega|\omega_l, \omega_h)$ is the frequency response of the stop band filter. The impulse response of this filter is

$$\begin{aligned} d(n|\omega_l, \omega_h) &= \delta(n) - f(n|\omega_l, \omega_h) && (3.14)\\ &= \delta(n) - \frac{2}{n\pi}\cos(\omega_1 n)\sin(\omega_2 n). && (3.15) \end{aligned}$$

3.1.2 Choice of Windows

Four types of windows are discussed here. They are the triangular, generalized Hamming, Kaiser, and Chebyshev windows. As was noted in the introduction it is the frequency response of a window which governs its efficacy in filter design. Consequently, for each window type, we try to give its frequency response and a qualitative analysis of its features with respect to the frequency response of the rectangular window.

The frequency response of the rectangular window is obtained by taking the Fourier transform of (3.4)

$$\begin{aligned} R_N(\omega) &= \sum_{n=0}^{N-1} e^{-j\omega n} && (3.16)\\ &= \frac{\sin(\omega N/2)}{\sin(\omega/2)} e^{-j(N-1)\omega/2}. && (3.17) \end{aligned}$$

The magnitude of (3.17) is plotted as the solid line in Figure ??. Evaluating (3.17) at $\omega = 0$ yields the height of the main lobe which is $R_N(0) = N$. The zeros of $R_N(\omega)$ are located at $\omega = \pm 2\pi n/N$, $n = 1, 2, \ldots$, and, consequently, the base of the main lobe has width $4\pi/N$. The area under the main lobe can be bounded from above by the area of a rectangle (depicted by a dotted curve in Figure ??) of area 4π and from below by that of a triangle (also shown in Figure ??) of area 2π. Thus, the area under the main lobe is essentially independent of the value of N and the percentage area under the main lobe decreases with increasing N. This fact is important because it illustrates that the rectangular window is limited in its ability to perform like an impulse.

`exec('fir6.code')` Magnitude of rectangular window f6

By comparison the percentage area under the main lobe of the triangular window is approximately constant as the value of N increases. The impulse response of the triangular window is

$$T_{2N-1}(n) = \begin{cases} (n+1)/N, & 0 \le n \le N-1 \\ (2N-1-n)/N, & N \le n \le 2N-2 \\ 0, & \text{otherwise.} \end{cases} \quad (3.18)$$

Since the impulse response for the triangular window can be obtained by scaling the rectangular window by $1/\sqrt{N}$ and convolving it with itself, the frequency response, $T_{2N-1}(\omega)$, is the square of $R_N(\omega)/N$ or

$$T_{2N-1}(\omega) = \frac{\sin^2(\omega N/2)}{N\sin^2(\omega/2)} e^{-j(N-1)\omega}. \quad (3.19)$$

As can be seen from (3.19), the width of the main lobe of the triangular window is the same width as that of the rectangular window (i.e. $4\pi/N$). However, the impulse response of the triangular window is twice as long as that of the rectangular window. Consequently, the triangularly windowed filter shows less ripple but broader transition bands than the rectangularly windowed filter.

The Hamming window is like the triangular window in that its main lobe is about twice as wide as that of a rectangular window for an equal length impulse response. All but .04% of the

Hamming windows energy is in the main lobe. The Hamming window is defined by

$$H_N(n) = \begin{cases} \alpha + (1-\alpha)\cos(\frac{2\pi n}{N}), & -(N-1)/2 \leq n \leq (N-1)/2 \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

where $\alpha = .54$. Other values for α are possible. For example when $\alpha = .5$ then (3.20) is known as the Hanning window.. The frequency response of (3.20) can be obtained by noting that $H_N(n)$ is a rectangularly windowed version of the constant α and an infinite length cosine. Thus

$$\begin{aligned} H_N(\omega) &= R_N(\omega) * & (3.21) \\ & \quad [\alpha\delta(\omega) + \frac{1}{2}(1-\alpha)\delta(\omega - \frac{2\pi}{N}) + \frac{1}{2}(1-\alpha)\delta(\omega + \frac{2\pi}{N})] & (3.22) \\ &= \alpha R_N(\omega) + (\frac{1-\alpha}{2})R_N(\omega + \frac{2\pi}{N}) + (\frac{1-\alpha}{2})R_N(\omega - \frac{2\pi}{N}). & (3.23) \end{aligned}$$

where the "$*$" symbol denotes convolution.

The Kaiser window is defined as

$$K_N(n) = \begin{cases} \frac{I_o(\beta\sqrt{1-[2n/(N-1)]^2})}{I_o(\beta)}, & -(N-1)/2 \leq n \leq (N-1)/2 \\ 0, & \text{otherwise.} \end{cases} \tag{3.24}$$

where $I_o(x)$ is the modified zeroth-order Bessel function and β is a constant which controls the trade-off of the side-lobe heights and the width of the main lobe. The Kaiser window yields an optimal window in the sense that the side lobe ripple is minimized in the least squares sense for a certain main lobe width. A closed form for the frequency response of the Kaiser window is not available.

The Chebyshev window is obtained as the inverse DFT of a Chebyshev polynomial evaluated at equally spaced intervals on the unit circle. The Chebyshev window uniformly minimizes the amount of ripple in the side lobes for a given main lobe width and filter length. A useful aspect of the design procedure for the Chebyshev window is that given any two of the three parameters: the window length, N; half the main lobe width, δ_f; the side lobe height, δ_p, the third can be determined analytically using the formulas which follow. For δ_f and δ_p known, N is obtained from

$$N \geq 1 + \frac{\cosh^{-1}((1+\delta_p)/(\delta_p))}{\cosh^{-1}(1/(\cos(\pi\delta_f)))}. \tag{3.25}$$

For N and δ_p known, δ_f is obtained from

$$\delta_f = \frac{1}{\pi}\cos^{-1}(1/\cosh(\cosh^{-1}((1+\delta_p)/\delta_p)/(N-1))). \tag{3.26}$$

Finally, for N and δ_f known, δ_p is obtained from

$$\delta_p = [\cosh((N-1)\cosh^{-1}(1/\cos(\pi\delta_f))) - 1]^{-1}. \tag{3.27}$$

3.1.3 How to use wfir

The syntax for the function `wfir` is as follows can take two formats. The first format is as follows:

```
--> [wft,wfm,fr]=wfir()
```

where the parentheses are a required part of the name. This format of the function is interactive and will prompt the user for required input parameters such as the filter type (lp='low pass', hp='high pass', bp='band pass', sb='stop band'), filter length (an integer $n > 2$), window type (re='rectangular', tr='triangular', hm='hamming', kr='kaiser', ch='chebyshev') and other special parameters such as α for the the generalized Hamming window ($0 < \alpha < 1$) and β for the Kaiser window ($\beta > 0$). The three returned arguments are:

- wft: A vector containing the windowed filter coefficients for a filter of length n.
- wfm: A vector of length 256 containing the frequency response of the windowed filter.
- fr: A vector of length 256 containing the frequency axis values ($0 \leq$ fr$\leq .5$) associated to the values contained in wfm.

The second format of the function is as follows:

```
--> [wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

This format of the function is not interactive and, consequently, all the input parameters must be passed as arguments to the function. The first argument `ftype` indicates the type of filter to be constructed and can take the values `'lp'`, `'hp'`, `'bp'`, and `sb'` representing, respectively the filters low-pass, high-pass, band-pass, and stop-band. The argument `forder` is a positive integer giving the order of the desired filter. The argument `cfreq` is a two-vector for which only the first element is used in the case of low-pass and high-pass filters. Under these circumstances `cfreq(1)` is the cut-off frequency (in normalized Hertz) of the desired filter. For band-pass and stop-band filters both elements of `cfreq` are used, the first being the low frequency cut-off and the second being the high frequency cut-off of the filter. Both values of `cfreq` must be in the range $[0, .5)$ corresponding to the possible values of a discrete frequency response. The argument `wtype` indicates the type of window desired and can take the values `'re'`, `'tr'`, `'hm'`, `'hn'`, `'kr'`, and `'ch'` representing, respectively, the windows rectangular, triangular, Hamming, Hanning, Kaiser, and Chebyshev. Finally, the argument `fpar` is a two-vector for which only the first element is used in the case of Kaiser window and for which both elements are used in the case of a Chebyshev window. In the case of a Kaiser window the first element of `fpar` indicates the relative trade-off between the main lobe of the window frequency response and the side-lobe height and must be a positive integer. For more on this parameter see [24]. For the case of the Chebyshev window one can specify either the width of the window's main lobe or the height of the window sidelobes. The first element of `fpar` indicates the side-lobe height and must take a value in the range $[0, 1)$ and the second element gives the main-lobe width and must take a value in the range $[0, .5)$. The unspecified element of the `fpar`-vector is indicated by assigning it a negative value. Thus, `fpar=[.01,-1]` means that the Chebyshev window will have side-lobes of height .01 and the main-lobe width is left unspecified.

Note: Because of the properties of FIR linear phase filters it is not possible to design an even length high pass or stop band filter.

3.1.4 Examples

This section gives several examples of windowed filter design. In the first example we choose a low pass filter of length $n = 33$ using a Kaiser window with parameter $\beta = 5.6$. The resulting magnitude of the windowed filter is plotted in Figure **??** where the magnitude axis is given on a log scale. `exec('fir7.code')` Low pass filter with Kaiser window, $n = 33$, $\beta = 5.6$ f7

The second example is a stop band filter of length 127 using a Hamming window with parameter $\alpha = .54$. The resulting magnitude of the windowed filter is plotted in Figure **??** where the magnitude is given on a log scale. `exec('fir8.code')` Stop band filter with Hamming window, $n = 127$, $\alpha = .54$ f8

The third example is a band pass filter of length 55 using a Chebyshev window with parameter $dp = .001$ and $df = .0446622$. The resulting magnitude of the windowed filter is plotted in Figure **??** where the magnitude is given on a log scale. `exec('fir9.code')` Band pass filter with Chebyshev window, $n = 55$, $dp = .001$, $df = .0446622$ f9

3.2 Frequency Sampling Technique

This technique is based on specification of a set of samples of the desired frequency response at N uniformly spaced points around the unit circle, where N is the filter length. The z-transform of an FIR filter is easily shown to be :

$$H(z) = \frac{1 - z^{-N}}{N} \sum_{k=0}^{N-1} \frac{H(k)}{(1 - z^{-1} e^{j(2\pi/N)k})} \tag{3.28}$$

This means that one way of approximating any continuous frequency response is to *sample in frequency*, at N equi-spaced points around the unit circle (the frequency samples), and interpolate between them to obtain the continuous frequency response. Thus, the approximation error will be exactly zero at the sampling frequencies and finite between them. This fact has to be related to the reconstruction of a continuous function from its samples, as exposed in section 2.2 for the case of a continuous-time signal.

The interpolation formula for an FIR filter, that is its frequency response, is obtained by evaluating (3.28) on the unit circle:

$$\begin{aligned} H(e^{j\omega}) &= \frac{e^{-j\omega(N-1)/2}}{N} \sum_{k=0}^{N-1} \frac{H(k)e^{-jk\pi/N}\sin(N\omega/2)}{\sin(\omega/2 - k\pi/N)} \\ &= \frac{e^{-j\omega(N-1)/2}}{N} \sum_{k=0}^{N-1} H(k)S(\omega,k) \end{aligned} \tag{3.29}$$

where

$$\begin{aligned} S(\omega,k) &= e^{-jk\pi/N} \frac{\sin(N\omega/2)}{\sin(\omega/2 - k\pi/N)} \\ &= \pm e^{-jk\pi/N} \frac{\sin(N(\omega/2) - k\pi/N)}{\sin(\omega/2 - k\pi/N)} \end{aligned} \tag{3.30}$$

are the interpolating functions. Thus, the contribution of every frequency sample to the continuous frequency response is proportional to the interpolating function $\sin(N\omega/2)/\sin(\omega/2)$ shifted by $k\pi/N$ in frequency. The main drawback of this technique is the lack of flexibility in specifying the transition band width, which is equal to the number of samples the user decides to put in times π/N, and thus is strongly related to N. Moreover, the specification of frequency samples in transition bands, giving minimum ripple near the band edges, is not immediate. Nevertheless, it will be seen, in a later chapter on filter optimization techniques, that simple linear programming techniques can be used to drastically reduce the error approximation by optimizing only those samples located in the transition bands. To illustrate this point, Figure **??** shows the response obtained for a type 1 band pass filter with length 65 : first with no sample in the transition bands and second (dashed curve) with one sample of magnitude .5 in each of these bands. It is worth noting at this point that the linear-FIR design problem with arbitrary frequency response specification is more efficiently solved using a minmax approximation approach, which is exposed in the next section. `exec('fstyp121.code')`Type 1 band pass filter with no sample or one sample in each transition bandf.fs1 Finally, depending on where the initial frequency sample occurs, two distinct sets of frequency samples can be given, corresponding to the so-called type 1 and type 2 FIR filters :

$$\begin{aligned} f_k &= \frac{k}{N} \; k = 0, \ldots, N-1 \text{ for type 1 filters} \\ f_k &= \frac{k+1/2}{N} \; k = 0, \ldots, N-1 \text{ for type 2 filters} \end{aligned}$$

The type of design is at user's will and depends on the application: for example, a band edge may be closer to a type 1 than to a type 2 frequency sampling point. This point is illustrated in Figure **??** for the case of a low pass filter with length 64 and no sample in the transition band. `exec('fstyp122.code')`Type 1 and type 2 low pass filterf.fs2 The full line (resp. the dashed line) gives the approximated response for the type 1 (resp. type 2) FIR linear filter. We give now the way the two previous examples have been generated and the code of the function `fsfir` which calculates the approximated response. Figure **??** was obtained with the following set of instructions :

and Figure **??** with :

3.3 Optimal filters

The design of FIR linear phase filters is a topic addressed in some detail in the section on windowed filter design. The essential idea behind the techniques of windowed filter design is to obtain a filter which is close to a minimum squared error approximation to the desired filter. This section is devoted to the description of a filter design function which seeks to optimize such an alternative criterion : the minimax or Chebyshev error approximation.

3.3.1 Minimax Approximation

To illustrate the problem of minimax approximation we propose an overspecified system of N linear equations in M unknowns where $N > M$. If x represents the unknown M-vector then the system of equations can be written as

$$Ax = b \tag{3.31}$$

where A is an $N \times M$ matrix and b is an N-vector. In general, no solution will exist for (3.31) and, consequently, it is reasonable to seek an approximation to x such that the error vector

$$r(x) = Ax - b \tag{3.32}$$

is in some way minimized with respect to x.

Representing the N components of $r(x)$ as $r_k(x)$, $k = 1, 2, \ldots, N$ the minimax approximation problem for the system of linear equations in (3.31) can be posed as follows. The minimax approximation, $\hat{x}_\infty$, is obtained by finding the solution to

$$\hat{x}_\infty = \arg\min_x ||r_k(x)||_\infty \tag{3.33}$$

where

$$||r_k(x)||_\infty = \max_k |r_k(x)|. \tag{3.34}$$

Equation (3.34) defines the supremum norm of the vector $r(x)$. The supremum norm of $r(x)$ for a particular value of x is the component of $r(x)$ (i.e., the $r_k(x)$) which is the largest. The minimax approximation in (3.33) is the value of x which, out of all possible values for x, makes (3.34) the smallest.

The minimax approximation can be contrasted by the minimum squared error approximation, $\hat{x}_2$, as defined by

$$\hat{x}_2 = \arg\min_x ||r(x)||_2 \tag{3.35}$$

where

$$||r(x)||_2 = [\sum_{k=1}^{N} {r_k}^2(x)]^{1/2}. \tag{3.36}$$

There is a relationship between (3.34) and (3.36) which can be seen by examining the class of norms defined on $r(x)$ by

$$||r(x)||_n = [\sum_{k=1}^{N} {r_k}^n(x)]^{1/n}. \tag{3.37}$$

For $n = 2$ the expression in (3.37) is the squared error norm in (3.36) and for $n \to \infty$ the norm in (3.37) becomes the supremum norm in (3.34) (which explains the notation $|| \cdot ||_\infty$). If $r(x)$ was a continuous function instead of a discrete component vector then the sum in (3.36) would become an integral and the interpretation of the approximation in (3.35) would be that the best approximation was the one which minimized the area under the magnitude of the error function $r(x)$. By contrast the interpretation of the approximation in (3.33) would be that the best approximation is the one which minimizes the maximum magnitude of $r(x)$.

As an example, consider the system of four linear equations in one unknown:

$$\begin{aligned} x &= 2 \\ \frac{1}{3}x &= 1 \\ x &= 4 \\ \frac{6}{15}x &= 3 \end{aligned} \tag{3.38}$$

The plot of the magnitude of the four error functions $|r_k(x)|$, $k = 1, 2, 3, 4$ is shown in Figure **??**. `exec('remez1.code')` Minimax Approximation for Linear Equations f5.01 Also shown in Figure **??** is a piece-wise continuous function denoted by the cross-hatched segments of the $r_k(x)$. This is the function which represents $||r(x)||_\infty$ as a function of x. Consequently, it is easy to see which value of x minimizes $||r(x)||_\infty$ for this problem. It is the value of x which lies at the cross-hatched intersection of the functions $|x - 2|$ and $|\frac{6}{15}x - 3|$, that is $\hat{x}_\infty = 3.571$. The maximum error at this value is 1.571.

By comparison the mean squared error approximation for a system of linear equations as in (3.31) is

$$\hat{x}_2 = (A^TA)^{-1}A^Tb \tag{3.39}$$

where T denotes the transpose (and assuming that A^TA is invertible). Consequently, for the example in (3.38) we have that $A = [1, \frac{1}{3}, 1, \frac{6}{15}]^T$ and that $b = [2, 1, 4, 3]^T$ and thus $\hat{x}_2 = 3.317$. The maximum error here is 1.673. As expected the maximum error for the approximation $\hat{x}_2$ is bigger than that for the approximation $\hat{x}_\infty$.

3.3.2 The Remez Algorithm

The Remez algorithm seeks to uniformly minimize the magnitude of an error function $E(f)$ on an interval $[f_0, f_1]$. In the following discussion the function $E(f)$ takes the form of a weighted difference of two functions

$$E(f) = W(f)(D(f) - H(f)) \tag{3.40}$$

where $D(f)$ is a single-valued function which is to be approximated by $H(f)$, and $W(f)$ is a positive weighting function. The Remez algorithm iteratively searches for the $H^*(f)$ such that

$$H^*(f) = \arg \min_{H(f)} \|E(f)\|_\infty \tag{3.41}$$

where

$$\|E(f)\|_\infty = \max_{f_0 \leq f \leq f_1} |E(f)| \tag{3.42}$$

is known as both the Chebyshev and the minimax norm of $E(f)$. The details of the Remez algorithm can be found in [5].

The function $H(f)$ is constrained, for our purposes, to the class of functions

$$H(f) = \sum_{n=0}^{N} a_n \cos(2\pi f n). \tag{3.43}$$

Furthermore, we take the interval of approximation to be $[0, .5]$. Under these conditions the posed problem corresponds to digital filter design where the functions $H(f)$ represent the discrete Fourier transform of an FIR, linear phase filter of odd length and even symmetry. Consequently, the function $H(f)$ can be written

$$H(f) = \sum_{n=-N}^{N} h_n e^{-j2\pi f n} \tag{3.44}$$

The relationship between the coefficients in (3.43) and (3.44) is $a_n = 2h_n$ for $n = 1, 2, \ldots, N$ and $a_0 = h_0$.

With respect to the discussion in the previous section the problem posed here can be viewed as an overspecified system of linear equations in the $N + 1$ unknowns, a_n, where the number of equations is uncountably infinite. The Remez algorithm seeks to solve this overspecified system of linear equations in the minimax sense. The next section describes the Scilab function `remezb` and how it can be used to design FIR filters.

3.3.3 Function `remezb`

The syntax for the function `remezb` is as follows:

```
--> an=remezb(nc,fg,df,wf)
```

where `df` and `wf` are vectors which are sampled values of the functions $D(f)$ and $W(f)$ (see the previous section for definition of notation), respectively. The sampled values of $D(f)$ and $W(f)$ are taken on a grid of points along the f-axis in the interval $[0, .5]$. The values of the frequency grid are in the vector `fg`. The values of `fg` are not obliged to be equi-spaced in the interval $[0, .5]$. In fact, it is very useful, for certain problems, to specify an `fg` which has elements which are equi-spaced in only certain sub-intervals of $[0, .5]$ (see the examples in the following section). The value of `nc` is the number of cosine functions to be used in the evaluation of the approximating function $H(f)$ (see (3.43)). The value of the variable `nc` must be a positive, odd integer if the problem is to correspond to an FIR filter. The `an` are the values of the coefficients in (3.43) which correspond to the optimal $H(f)$.

To obtain the coefficients of the corresponding FIR filter it suffices to create a vector `hn` using the Scilab commands:

```
//exec('remez8.code')
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

Even length filters can be implemented as follows. For an even length filter to have linear phase the filter must have even symmetry about the origin. Consequently, it follows that the filter must take values at the points $n = \pm\frac{1}{2}, \pm\frac{3}{2}, \ldots, \pm\frac{N-1}{2}$ and that the frequency response of the filter has the form

$$H(f) = \sum_{n=-N-\frac{1}{2}}^{N+\frac{1}{2}} h_n e^{-j2\pi fn}. \tag{3.45}$$

Due to the even symmetry of the frequency response, $H(f)$, (3.45) can be put into the form

$$H(f) = \sum_{n=1}^{N} b_n \cos[2\pi(n-\frac{1}{2})f] \tag{3.46}$$

where the relationship between the h_n in (3.45) and the b_n in (3.46) is $h(n) = \frac{1}{2}b(N-n)$ for $n = 1, 2, \ldots, N$.

The expression for $H(f)$ in (3.46) can be rewritten so that

$$H(f) = \cos(\pi f) \sum_{n=0}^{N-1} \tilde{b}_n \cos(2\pi nf). \tag{3.47}$$

where $b(n) = \frac{1}{2}[\tilde{b}(n-1)+\tilde{b}(n)]$ for $n = 2, 3, \ldots, N-1$ and $b(1) = \tilde{b}(0)+\frac{1}{2}\tilde{b}(1)$ and $b(N) = \frac{1}{2}\tilde{b}(N-1)$. Since the expression in (3.47) is identical to that in (3.43) all that is required to make the function `remezb` work is a change in the values of the desired and weight vectors by the factor $\cos^{-1}(\pi f)$. That is, the arguments given to the function `remezb` are `ddf` and `wwf` where `ddf` $=$ `df`$/\cos(\pi f)$ and `wwf` $=$ `wf`$\cos(\pi f)$. Caution must be used in choosing the values of `fg` since for $f = .5$ the division of `df` by $\cos(\pi f) = 0$ is not acceptable. The output, `an`, of the function can be converted to the filter coefficients `hn` by using the Scilab commands

```
//exec('remez2.code')
hn(1)=.25*an(nc);
hn(2:nc-1)=.25*(an(nc:-1:3)+an(nc-1:-1:2));
hn(nc)=.5*an(1)+.25*an(2);
hn(nc+1:2*nc)=hn(nc:-1:1);
```

Noting that the form of (3.47) has the term $\cos(\pi f)$ as a factor, it can be seen that $H(.5) = 0$ regardless of the choice of filter coefficients. Consequently, the user should not attempt to design filters which are of even length and which have non-zero magnitude at $f = .5$.

3.3.4 Examples Using the function `remezb`

Several examples are presented in this section. These examples show the capabilities and properties of the function `remezb`. The first example is that of a low-pass filter with cut-off frequency .25. The number of cosine functions used is 21. The input data to the function are first created and then passed to the function `remezb`. The subsequent output of cosine coefficients is displayed below.

Notice that the frequency grid `fg` is a vector of frequency values which are equally spaced in the interval $[0, .5]$. The desired function `ds` is a vector of the same length as `fg` and which takes the value 1 in the interval $[0, .25]$ and the value 0 in $(.25, .5]$. The weight function `wt` is unity for all values in the interval.

As described in the previous section the cosine coefficients `an` are converted into the coefficients for a even symmetry FIR filter which has frequency response as illustrated in Figure ??. `exec('remez2_4.code')` Low Pass Filter with No Transition Band f5.1

The error of the solution illustrated in Figure 3.13 is very large; it can become reasonable by leaving a transition band when giving the specification of the frequency grid. The following example shows how this is done; `remezb` is specified as follows :

Here the frequency grid `fg` is specified in the intervals $[0, .24]$ and $[.26, .5]$ leaving the interval $[.24, .26]$ as an unconstrained transition band. The frequency magnitude response of the resulting filter is illustrated in Figure ??. As can be seen the response in Figure ?? is much more acceptable than that in Figure ??. `exec('remez2_4.code')` Low Pass Filter with Transition Band $[.24, .26]$ f5.2

A third and final example using the function `remezb` is illustrated below. In this example the desired function is triangular in shape. The input data was created using the following Scilab commands

The resulting frequency magnitude response is illustrated in Figure ??.This example illustrates the strength of the function `remezb`. The function is not constrained to standard filter design problems such as the class of band pass filters. The function is capable of designing linear phase FIR filters of any desired magnitude response.

`exec('remez2_4.code')` Triangular Shaped Filter f5.3

3.3.5 Scilab function `eqfir`

For the design of piece-wise constant filters (such as band pass, low pass, high pass, and stop band filters) with even or odd length the user may use the function `eqfir` which is of simpler manipulation. Three examples are presented here. The first two examples are designs for a stopband filter. The third example is for a design of a high pass filter.

The first design for the stop band filter uses the following Scilab commands to create the input to the function `eqfir`:

The resulting magnitude response of the filter coefficients is shown in Figure **??**. As can be seen the design is very bad. This is due to the fact that the design is made with an even length filter and at the same time requires that the frequency response at $f = .5$ be non-zero.

`exec('remez5_7.code')` Stop Band Filter of Even Length f5.5

The same example with `nf` $= 33$ is run with the result shown in Figure **??**. `exec('remez5_7.code')` Stop Band Filter of Odd Length f5.6

The final example is that of a high pass filter whose input parameters were created as follows:

The result is displayed in Figure **??**. `exec('remez5_7.code')` High Pass Filter Design f5.7

Chapter 4

Design of Infinite Impulse Response Filters

4.1 Analog filters

In this section we review some of the most classical analog (or "continuous time") filters. These are defined in frequency domain as rational transfer functions of the form:

$$H(s) = \frac{\sum_{i=0}^{m} b_i s^i}{1 + \sum_{i=1}^{n} a_i s^i}$$

The problem is then to determine the a_i and b_i coefficients or equivalently the zeros z_i and poles p_i of H in order to meet given specifications of the (squared) magnitude response defined as:

$$h^2(\omega) = |H(i\omega)|^2 = H(s)H(-s)|_{s=i\omega} \tag{4.1}$$

Thus, $h(\omega)$ is the spectrum of the output of a linear filter which admits a white noise as input. We shall consider in this section only prototype lowpass filters, i.e., ideally we want to obtain $h(\omega) = 0$ for $\omega > \omega_c$ and $h(\omega) = 1$ for $\omega < \omega_c$. Highpass, bandpass and stopband filters are then easily constructed by a simple change of variables.

The construction consists of finding a function H_2 of the complex variable s such that $H_2(s) = H(s)H(-s)$ (i.e., is symmetric with respect to the imaginary axis and such that $H_2(i\omega) = h^2(\omega)$ along the imaginary axis). Furthermore, the function $H_2(s)$ will be chosen to be rational and, consequently, defined by its poles and zeros.

The transfer function of the filter, $H(s)$, will then be defined by selecting all the poles and zeros which lie in the left hand side of the complex s-plane. In this way we obtain a stable and minimum phase filter.

4.1.1 Butterworth Filters

The squared-magnitude response that we want to realize is given by:

$$h_n^2(\omega|\omega_c) = \frac{1}{1 + (\frac{\omega}{\omega_c})^{2n}} \tag{4.2}$$

Here, ω_c is the cutoff frequency and n the order. A typical response can be plotted with the function `buttmag` (see Figure **??**):

The following code gives an example of the squared magnitude of a Butterworth filter of order 13 (see Figure **??**).

`exec('analog1.code')` Magnitude in dB. $n=13, \omega_c=300$ butt

From Figure **??** we see that the magnitude is a decreasing function of the frequency. Also we see that

$$h_n^2(\omega_c|\omega_c)=\frac{1}{2}$$

independently of the order n.

Let us define now the stable transfer function $H(s)$ (Butterworth filter) by:

$$H(s)=\frac{k_0}{\prod_{k=1}^n (s-p_k)}$$

where

$$p_k=e^{i\pi[1/2+(2k-1)/2n]} \qquad k=1,\ldots,n$$

This is done by the small function `zpbutt` which computes the poles p_k and the gain k_0:

For instance, with n=7 and $\omega_c=3$, we obtain the following transfer function:

The poles p_k of H are located on the unit circle and are symmetric with respect to the real axis as is illustrated in Figure **??**. The figure was obtained as follows:

`exec('analog2.code')` Butterworth filter: pole positions. $n=13$ polesp

We note the symmetry of the coefficients in $H(s)$ i.e., that $H(s)=\tilde{H}(s)=s^n\,H(\frac{1}{s})$, which follows from the fact that for each pole p_k there corresponds the pole $\dfrac{1}{p_k}=\overline{p_k}$. Also, we see that $H(-s)$ is obtained in the same way as $H(s)$ by selecting the (unstable) poles $-p_k$ instead of the p_k. Since the set $\{(p_k,-p_k) \quad k=1,\ldots,n\}$ is made with the $2n$ roots of the polynomial $p(s)=1+(-s^2)^n$. Thus, we have:

$$H(s)H(-s)=\frac{1}{1+(-s^2)^n}$$

It follows immediately from (4.1) that $H(s)$ realizes the response (4.2).We see that $H(s)$ is obtained by a very simple spectral factorization of $p(s)$, which here can be done analytically.

Order determination: The filter order, n, completely specifies a Butterworth filter. In general n is determined by giving a desired attenuation $\frac{1}{A}$ at a specified "normalized" frequency $f=\frac{\omega_r}{\omega_c}$. The filter order, n, is given by the solution of $\frac{1}{A^2}=h_n^2(f)$. We obtain immediately:

$$n=\frac{\log_{10}(A^2-1)}{2\,\log_{10}(f)} \tag{4.3}$$

4.1.2 Chebyshev filters

The n^{th} order Chebyshev polynomial $T_n(x)$ is defined recursively by:

$$\begin{cases} T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) & \\ T_0(x) = 1 & T_1(x) = x \end{cases}$$

It may be shown that $T_n(x)$ is given more explicitly by:

$$T_n(x) = \begin{cases} \cos(n\cos^{-1}(x)) & \text{if } |x| < 1 \\ \cosh(n\cosh^{-1}(x)) & \text{otherwise} \end{cases} \tag{4.4}$$

The recursive function `chepol` implements this computation: We note that the roots of T_n are real and symmetric with respect to the imaginary axis. These polynomials are used to analytically define the squared-magnitude of the frequency response for a class of analog filters.

Type 1: Ripple in the passband

For type I Chebyshev filters the squared-magnitude function that we want to realize is:

$$h^2_{1,n}(\omega \mid \omega_c, \epsilon) = \frac{1}{1 + \epsilon^2 T^2_n(\frac{\omega}{\omega_c})} \tag{4.5}$$

This function is completely specified once its three parameters (ω_c, ϵ, n) are given.

Using `chepol`, it is easy to compute the squared-magnitude response (4.5). The function `cheb1mag` evaluates $h^2_{1,n}(\omega \mid \omega_c, \epsilon)$ for a given sample vector of $\omega's$. Note that for any value of n one has

$$h^2_{1,n}(\omega_c \mid \omega_c, \epsilon) = \frac{1}{1 + \epsilon^2}$$

The function h_1 is decreasing for $\omega > \omega_c$ with "fast" convergence to zero for "large" values of the order n. The number of oscillations in the passband is also increasing with n. If at $\omega = \omega_r > \omega_c$, $h^2_{1,n}$ reaches the value $\frac{1}{A^2}$ then the parameters (ω_c, ϵ, n) and (A, ω_r) are linked by the equation $h^2_{1,n}(\omega_r|\omega_c, \epsilon) = \frac{1}{A^2}$ which may be written also as

$$A^2 = 1 + \epsilon^2 T^2_n(\frac{\omega_r}{\omega_c}) \tag{4.6}$$

Using (4.4) this latter equation may be solved more explicitly: $n\cosh^{-1}(f) = \cosh^{-1}(g)$ with $f = \dfrac{\omega_r}{\omega_c}$ and $g = \frac{A^2-1}{\epsilon}$.

Below is an example of the magnitude plotted in Figure **??**.

`exec('analog3.code')` Magnitude of a Type 1 Chebyshev filter ch1mg Let us now calculate the transfer function of a Type 1 Chebyshev filter. The transfer function is all-pole and the poles lie on an ellipse which can be determined totally by specifying the parameter ϵ, the order n and the cutoff frequency ω_c. The horizontal and vertical rays a and b of this ellipse are given by: $a = \omega_c \dfrac{\gamma - \gamma^{-1}}{2}$ and $b = \omega_c \dfrac{\gamma + \gamma^{-1}}{2}$ where

$$\gamma = (\frac{1 + \sqrt{1 + \epsilon^2}}{\epsilon})^{1/n}$$

The poles ($p_k = \sigma_k + i\,\Omega_k, k = 1, 2, \ldots, n$) are simple and, in order to have a stable filter, are regularly spaced over the left hand side of the ellipse. The function `zpch1` computes the poles (and the gain) of a Type 1 Chebyshev filter.

With the function `zpch1` we can now calculate the transfer function which realizes the preceding example and recover the desired magnitude. Compare Figures **??** and **??**, the latter figure being obtained as follows:

`exec('analog4.code')` Chebyshev filter: frequency response in magnitudech1mg2

Order determination: We have seen that the parameter ϵ specifies the size of the passband ripple and ω_c is the passband edge, thus, one has

$$\frac{1}{1+\epsilon^2} \leq h_{1,n}^2(\omega \mid \omega_c) \leq 1 \qquad \text{for} \quad 0 \leq \omega \leq \omega_c$$

The order n is generally determined by specifying a desired attenuation $\frac{1}{A}$ at a given "normalized" frequency $f = \frac{\omega_r}{\omega_c}$. As in (4.6), n is given by the solution of $\frac{1}{A^2} = h_{1,n}^2(f|\omega_c, \epsilon)$:

$$n = \frac{\cosh^{-1}(\frac{\sqrt{A^2-1}}{\epsilon})}{\cosh^{-1}(f)} = \frac{\log(g + \sqrt{(}g^2 - 1))}{\log(f + \sqrt{(}f^2 - 1)}$$

where $g = \sqrt{(}\frac{A^2-1}{\epsilon^2})$

Type 2: Ripple in the stopband

The squared-magnitude response of a Type 2 Chebyshev filter is defined by:

$$h_{2,n}^2(\omega \mid \omega_r, A) = \frac{1}{1 + \frac{A^2-1}{T_n^2(\frac{\omega_r}{\omega})}}$$

Here ω_r is the passband edge and A the attenuation at ω_r. The function `cheb2mag` computes the squared-magnitude response. Note that the sample vector must not include the value zero. Also, for any value of the order n one has:

$$h_{2,n}^2(\omega_r \mid \omega_r, A) = \frac{1}{A^2}$$

The function is decreasing for $0 < \omega < \omega_r$ and displays ripple for $\omega > \omega_r$.

Also note that when the equation (4.6) is satisfied both Type 1 and type 2 functions $h_{1,n}$ and $h_{2,n}$ take the same values at ω_c and ω_r:

$$h_{1,n}^2(\omega_r|\omega_c, \epsilon) = h_{2,n}^2(\omega_r|\omega_r, A) = \frac{1}{A^2}$$

$$h_{2,n}^2(\omega_c|\omega_c, \epsilon) = h_{2,n}^2(\omega_c|\omega_r, A) = \frac{1}{1+\epsilon^2}$$

We can now plot for example a squared-magnitude response for the following specifications: $1/A = 0.2, \omega_r = 6$, and $n = 10$. The sample vector of ω's is chosen as `0:0.05:10`. The magnitude plotted in dB is given by Figure **??**, and was generated by the following code:

`exec('analog5.code')` Magnitude of a Type 2 Chebyshev filter ch2mg

The transfer function of a type 2 Chebyshev filter has both poles and zeros. The zeros are imaginary and are located at

$$z_k = i\frac{\omega_r}{\cos(\frac{(2k-1)\pi}{2n})} \qquad k = 1, 2, \ldots, n$$

The poles $p_k = \sigma_k + i\Omega_k \quad k = 1, 2, \ldots, n$ are found by solving for the singularities of the denominator of h. They are defined by:

$$\sigma_k = \frac{\alpha_k}{{\alpha_k}^2 + {\beta_k}^2}$$

$$\Omega_k = \frac{-\beta_k}{{\alpha_k}^2 + {\beta_k}^2}$$

where

$$\alpha_k = -a \ \sin(\frac{(2k-1)\pi}{2n})$$

$$\beta_k = b \ \cos(\frac{(2k-1)\pi}{2n})$$

and

$$a = \frac{\gamma - \gamma^{-1}}{2}$$

$$b = \frac{\gamma + \gamma^{-1}}{2}$$

$$\gamma = (A + \sqrt{A^2 - 1})^{1/n}$$

The function `zpch2` computes the poles and zeros of a type 2 Chebyshev filter, given its parameters (ω_r, A, n), according to the preceding formulas.

Let us consider the preceding example: we had $n = 10$, $\omega_r = 6$, $A = 5$.

Order determination: We have seen that the parameter A specifies the size of the passband ripple and ω_r is the stopband edge. Thus, one has

$$0 \leq h^2_{2,n}(\omega \mid \omega_r, A) \leq \frac{1}{A^2} \qquad \text{for} \quad \omega_r \leq \omega$$

The order n is generally determined by specifying a desired attenuation $\frac{1}{1+\epsilon^2}$ at a given "normalized" frequency $f = \frac{\omega_r}{\omega_c}$ (remember that in order to define a type 2 Chebyshev filter ω_r must be given). In a similar way as in (4.6), n is given by the solution of $\frac{1}{1+\epsilon^2} = h^2_{2,n}(f|\omega_r, A)$. Because of the symmetry in ϵ and A we obtain the same solution as for type 1 filters:

$$n = \frac{\cosh^{-1}(\frac{\sqrt{A^2-1}}{\epsilon})}{\cosh^{-1}(f)} = \frac{\log(g + \sqrt{(}g^2 - 1))}{\log(f + \sqrt{(}f^2 - 1)}$$

where $g = \sqrt{\dfrac{A^2 - 1}{\epsilon^2}}$

4.1.3 Elliptic filters

The elliptic filter presents ripple in both the passband and stopband. It is based on the properties of the Jacobian elliptic function (see [8],[1]) that we briefly introduce now.

Elliptic integral

Let us define for z in the complex plane the function

$$u(z) = \int_0^z \frac{dt}{(1-t^2)^{1/2}(1-mt^2)^{1/2}} \tag{4.7}$$

where m is a real constant $0 < m < 1$. (We will also use the notation $u(z, m)$ when necessary.)

We may assume that the functions $u_1 = (1-t^2)^{\frac{1}{2}}$ and $u_2 = (1-mt^2)^{\frac{1}{2}}$ are defined e.g. in the domain $\mathcal{D}$ made of the complex plane cut along the lines $\{z; Re(z) = \mp 1 \text{ and } Im(z) < 0\}$ and $\{z; Re(z) = \mp\frac{1}{\sqrt{m}} \text{ and } Im(z) < 0\}$. In other words we may choose for these complex functions the determination of the phase between $-\pi/2$ and $3\pi/2$. These functions are then completely specified in $\mathcal{D}$ by analytic continuation once we have fixed their values at 0 as being $+1$.

Let us define now in the above open connected domain $\mathcal{D}$ the function

$$\Phi(t) = \frac{1}{u_1(t)u_2(t)} = \frac{1}{(1-t^2)^{\frac{1}{2}}(1-mt^2)^{\frac{1}{2}}}$$

and consider the path $\mathcal{S}$ which encircles the positive right quarter of the complex plane composed of the positive real axis, the point at infinity and the imaginary axis traversed from ∞ to zero.

As t increases along the positive real axis the function $\Phi(t)$ first assumes positive real values for $0 \le t < 1$, then for $1 < t < 1/\sqrt{m}$, $\Phi(t)$ assumes purely imaginary values and for $t > 1/\sqrt{m}$, $\Phi(t)$ assumes real negative values. At ∞ we have $\Phi(t) = 0$ and as t decreases from ∞ to 0 along the imaginary axis $\Phi(t)$ assumes purely imaginary values.

Let us examine now the consequences of these properties on the function u. When z traces the same path $\mathcal{S}$ the function $u(z)$ traces the border of a *rectangle* $\mathcal{R}_0$ with corners at $(O, K, K + iK', iK')$, each side being the image of the four parts of the strip $\mathcal{S}$ defined above. This follows from the identities:

$$K(m) = \int_0^1 \Phi(t)dt = -\int_{\frac{1}{\sqrt{m}}}^{\infty} \Phi(t)dt$$

and

$$iK'(m) = \int_1^{\frac{1}{\sqrt{m}}} \Phi(t)dt = \int_0^{\infty} \Phi(it)dt$$

Thus the points $(0, 1, \frac{1}{\sqrt{(m)}}, \infty)$ of the real axis are respectively mapped by u into the points $(0, K, K + iK', iK')$ of the complex plane and the intervals $(0, 1)$,$(1, \frac{1}{\sqrt{m}})$,$(\frac{1}{\sqrt{m}}, \infty)$ are respectively mapped into the intervals $(0, K)$,$(K, K + iK')$,$(K + iK', iK')$.

It may be shown that u realizes a conformal mapping of the first quadrant of the complex plane into the rectangle $\mathcal{R}_0$. In particular any point z with $Re(z) \ge 0$ and $Im(z) \ge 0$ admits an unique image in the rectangle. In a similar way we can map, under u, each of the four quadrants of the complex plane into a rectangle $\mathcal{R}$ (called the "fundamental rectangle of periods") with corners at $(-K - iK', K - iK', K + iK', -K + iK')$, made of four smaller rectangles defined as $\mathcal{R}_0$ above and having the origin as common point. The function u defines a one-to-one correspondence of the complex plane into $\mathcal{R}$.

The function u has been implemented as the `%asn` function. This function may receive a real vector `x` as argument. The calculation is made componentwise. A specific algorithm [4] is implemented in fortran. We note that it sufficient to be able to compute $u(x)$ for $x \in (0,1)$ to have $u(z)$ for all nonnegative real and purely imaginary values z thanks to the following changes of variable:

$$\int_1^x \frac{dt}{(t^2-1)^{1/2}(1-mt^2)^{1/2}} = \int_0^y \frac{dt}{(1-t^2)^{1/2}(1-m_1t^2)^{1/2}}$$

with $m_1 = 1-m$, $y^2 = \frac{1}{m_1}\frac{x^2-1}{x^2}$ and $x \in (1, 1/\sqrt{m})$

$$\int_{\frac{1}{\sqrt{m}}}^x \frac{dt}{(t^2-1)^{1/2}(mt^2-1)^{1/2}} = \int_0^y \frac{dt}{(1-t^2)^{1/2}(1-mt^2)^{1/2}}$$

with $y^2 = \frac{1}{mx^2}$ and $x \in (1/\sqrt{m}, \infty)$

$$\int_0^x \frac{dt}{(1+t^2)^{1/2}(1+mt^2)^{1/2}} = \int_0^y \frac{dt}{(1-t^2)^{1/2}(1-m_1t^2)^{1/2}}$$

with $m_1 = 1-m$ and $y^2 = \frac{x^2}{1+x^2}$, which gives u for purely imaginary values of its argument.

We can now for example use `%asn` to plot three sides of the rectangle $\mathcal{R}_0$ as the image by `%asn` of the three associated real intervals (see Figure **??**). Here $m = 0.8$ and we see that how a linear scale is transformed by `%asn` (parametric plot). In particular at $x = 10$ the point iK' is not reached (as expected).

`exec('analog6.code')` The rectangle $\mathcal{R}_0$, image by u of the positive real axis.asn

The integrals K and K' are known as the "complete" elliptic integral: they may be calculated by `%asn` since we have $K(m) + iK'(m) = u(\frac{1}{\sqrt{(m)}}, m)$. These numbers may also be calculated by the Arithmetic-Geometric-Mean algorithm which is implemented as the `%K` function. Note that here m can be vector-valued and `%K` computes $K(m)$ componentwise (this fact will be useful later).

Elliptic function

For y inside the fundamental rectangle $\mathcal{R}$ the Jacobian elliptic function sn is defined as the inverse function of u i.e. we have for a fixed value of the parameter m:

$$u(z) = y \Leftrightarrow z = sn(y)$$

In particular we have for the corners of the rectangle $\mathcal{R}_0$ defined above: $sn(0) = 0$, $sn(K) = 1$, $sn(K+iK') = \frac{1}{\sqrt{m}}$, $sn(iK') = \infty$. In fact, the function sn may be extended to the full complex plane as a meromorphic function.

Indeed,the "symmetry principle" states that if f is an analytic function defined in an open set $\mathcal{D}$ whose boundary contains an *interval* L of the complex plane and if in addition this interval is itself mapped by f into another *interval* $L' = f(L)$ of the complex plane, then f can be extended to the set $\sigma(\mathcal{D})$, symmetric set of $\mathcal{D}$ with respect to L, by the formula $f(\sigma(z)) = \sigma'(f(z))$ where σ and σ' are the symmetries with respect to L and L' respectively.

Since the sides of the fundamental rectangle are mapped into intervals of the real and imaginary axis of the complex plane by sn, it may be easily shown that the function sn may be extended to the full complex plane by successive symmetries leading to a doubly periodic function with periods $4K$ and $2iK'$.

For real values of its argument, $sn(y)$ "behaves" like the sine function and for purely imaginary values of its argument, $sn(y)$ is purely imaginary and has poles at $\ldots, -3iK', -iK', iK', 3iK', \ldots$. For y in the interval $(-iK', +iK')$, $sn(y)$ "behaves" like i times the tan function and this pattern is repeated periodically. For $m = 0$, one has $K = \pi/2$, $K' = \infty$ and sn coincides with the sin function. For $m = 1$, one has $K = \infty$, $K' = \pi/2$ and sn coincides with the tanh function.

The function sn has been implemented by the following function `%sn` which calls a fortran routine for real values of the argument, and use the addition formulas ([1]) for imaginary values of the argument. Note that x is allowed to be complex and vector-valued.

Let us plot for example the real and imaginary behavior of sn; this is done by the following commands which produce the Figures **??** and **??** and give respectively $sn(x)$ for $0 \leq x \leq 4K$ and $sn(iy)$ for $0 \leq y \leq 3K'/2$.

`exec('analog7.code')` Behavior of the sn function for real valuesrealsn

For imaginary values of the argument we must take care of the pole of $sn(z)$ at $z = iK'$:

`exec('analog8.code')` Behavior of the sn function for imaginary valuesimagsn

Squared Magnitude Response of Elliptic Filter

The positions of poles and zeros of the sn function will allow to define the squared magnitude response of a prototype lowpass elliptic filter. The zeros of the sn function are located at $2pK+2qiK'$,where p and q are arbitrary integers and its poles are located at $2pK + (2q + 1)K'$.

For a fixed value of the parameter $m = m_1$, let us consider a path Σ_n joining the points $(0, nK_1, nK_1 + iK'_1, iK'_1)$ with n an *odd* integer and denoting $K_1 = K(m_1)$ and $K'_1 = K(1 - m_1)$. From the discussion above we see that for $z \in (0, nK_1)$ the function $sn(z)$ oscillates between 0 and 1 periodically as shown in Figure **??**. For $z \in (nK_1, nK + iK'_1)$, $sn(z)$ assumes purely imaginary values and increases in magnitude, with (real) limit values $sn(nK_1) = 1$ and $sn(nK_1+iK'_1) = \frac{1}{\sqrt{m_1}}$. Finally for $z \in (nK + iK_1', iK_1')$, $sn(z)$ oscillates periodically between $sn(nK_1 + iK_1') = \frac{1}{\sqrt{m_1}}$ and ∞.

For z in the contour Σ_n, let us consider now the function:

$$v(z) = \frac{1}{1 + \epsilon^2 sn^2(z, m_1)} \tag{4.8}$$

Clearly, $v(z)$ oscillates between 1 and $\frac{1}{1+\epsilon^2}$ for $z \in (0, nK_1)$ and between 0 and $\frac{1}{1+\frac{\epsilon^2}{m_1}}$ for $z \in (nK_1 + iK_1', iK_1')$. Also, clearly $v(z)$ is a continuous function of z, which is real-valued for z in the path Σ_n and if we chose the parameter $m_1 = \frac{\epsilon^2}{A^2-1}$ we can obtain an interesting behavior. The function `ell1mag` computes $v(z)$ for a given sample vector z in the complex plane and for given parameters ϵ and m_1.

Now, we define the vector $z = [z_1, z_2, z_3]$ as a discretization of the path Σ_n, with z_1 a dicretization of $(0, nK_1)$, z_2 a discretization of $(nK_1, nK_1 + ik_1')$ and z_3 a discretization of $(nK_1 + iK_1', iK_1')$. Then we can produce Figure **??** which clearly shows the behavior of $v(z)$ for the above three parts of z.

`exec('analog9.code')` v(z) for z in Σ_n, with $n = 9$riri

Of course, many other paths than Σ_n are possible, giving a large variety of different behaviors. Note that the number of oscillations in both the passband and stopband is n, an arbitrary odd integer and the amplitudes of the oscillations are entirely specified by the choice of the parameter m_1.

Thus, we have obtained a function v whose behavior seems to correspond to a "good" squared magnitude for the prototype filter that we want to realize. However, this function is defined along the path Σ_n of the complex plane. Since frequencies, ω, are given as positive real values we must find a mapping of the positive real axis onto the path Σ_n. But this is precisely done by the function $u(z) = sn^{-1}(z)$ given in (4.7) or more generally, after scaling by the function $\alpha sn^{-1}(\frac{z}{\omega_c}, m) + \beta$. Indeed, as we have seen, such a function maps the positive real axis into the border of a rectangle $\mathcal{R}(\alpha, \beta, m)$. The size of this rectangle depends on the parameter m and we have to chose m, α, β such that $\mathcal{R}(\alpha, \beta, m) = \Sigma_n$.

To be more specific, we have to choose now the value of the parameter m such that:

$$\alpha\, sn(nK_1, m) + \beta \quad = \quad \omega_c \tag{4.9}$$

$$\alpha\, sn(nK_1 + iK'_1, m) + \beta \quad = \quad \omega_r \tag{4.10}$$

Recalling that $sn^{-1}(1, m) = K(m)$ and $sn^{-1}(\frac{1}{\sqrt{m}}, m) = K(m) + iK'(m)$ we see that to satisfy these equations we must chose

$$\alpha = \frac{nK_1}{K} = \frac{K_1{}'}{K'}$$

and

$$\beta = \begin{cases} 0 & \text{if n is odd} \\ K_1 & \text{if n is even} \end{cases}$$

In particular, we obtain that the parameters n, $m_1 = \frac{\epsilon^2}{A^2-1}$ and $m = \frac{\omega_c{}^2}{\omega_r{}^2}$ of the filter cannot be chosen independently but that they must satisfy the equation:

$$n = \frac{K'(m_1)}{K(m_1)} \frac{K(m)}{K'(m)} = \frac{\chi_1}{\chi} \tag{4.11}$$

(We note $\chi_1 = \frac{K'(m_1)}{K(m_1)} = \frac{K(1-m_1)}{K(m_1)}$ and $\chi = \frac{K'(m)}{K(m)} = \frac{K(1-m)}{K(m)}$). Usually m_1 is "small" (close to 0) which yields χ_1 "large", and m is "large" (close to 1) which yields χ "large".

In practice very good specifications are obtained with rather low orders. In order to plot the frequency response magnitude of an elliptic prototype lowpass filter we can proceed as follows: first select the ripple parameters ϵ and A and compute $m_1 = \frac{\epsilon^2}{A^2-1}$ and $\chi_1 = \frac{K_1'}{K1}$, then for various integer values of n compute m such that equation (4.11) is satisfied or until the ratio $\frac{\omega_r}{\omega_c}$ is acceptable.

See Figure **??**

`exec('analog10.code')` log(m) versus log(m_1) for order n fixedisom.code

Much flexibility is allowed in the choice of the parameters, provided that equation (4.11) is satisfied. The functions `find_freq` and `find_ripple` may be used to find the stopband edge ω_r when ω_c, ϵ, A, and n are given and to find ϵ when the parameters n, ω_c, ω_r, and A are given.

The following code shows how to find compatible parameters and produce Figure **??**.

`exec('analog11.code')` Response of Prototype Elliptic Filter ellmag.code

Construction of the elliptic filter

The design of an elliptic filter is more complex than for the filters that we have defined until now. First the parameters n, m_1, and m which characterize the squared magnitude response cannot be chosen independently (see (4.11)). We have seen how to solve this difficulty. Second, the squared magnitude response is not a rational function and moreover it has an infinite number of poles and zeros.

The construction is accomplished in two steps: first a transformation is made in the complex plane which maps the real axis to the imaginary axis and transforms the rectangular path Σ_n to a rectangular path Σ'_n in the LHS of the complex plane. Then the elliptic function sn is used to perform a transformation which maps the imaginary axis into Σ'_n. Finally, only poles and zeros which are inside Σ'_n are kept for the transfer function.

Let us consider the pole-zero pattern of the function $v(z)$. Clearly, the poles of $sn(z)$ become double zeros of $v(z)$ and the poles of $v(z)$ are found by solving the equation:

$$1 + \epsilon^2\, sn^2(z) = 0$$

Thus the zeros of $v(z)$ in Σ_n are located at $i\,K', i\,K' + 2K, i\,K' + 4K, \ldots, i\,K' + 2p\,K$ and the poles of $v(z)$ in Σ_n are located at $i\,u_0, i\,u_0 + 2K, i\,u_0 + 4K, \ldots, i\,u_0 + 2p\,K$ with $2p + 1 = n$ and where we have noted $u_0 = sn^{-1}(\frac{i}{\epsilon}, m_1)$.

Consider now the transformation $\lambda = i\,\frac{K'(m)}{K'(m_1)}u = i\,\frac{K(m)}{nK(m_1)}u$ (n being given in (4.11)). The above pole-zero pole pattern is mapped inside the LHS of the complex plane and the contour Σ_n is mapped into $\Sigma'_n = (0, i\,K, -i\,K' + K, -K')$, and these points are respectively the image of the points $(0, i\,\omega_c, i\,\omega_r, i\,\infty)$ of the imaginary axis by the function $z \to i\,\omega_c\, sn(-iz, m)$.

The function `zpell` performs these transformations and computes the poles and zeros of a prototype elliptic filter.

We illustrate the use of this function by the following example which uses the preceding numerical values of the parameters ϵ, A, ω_c, ω_r. The code produces Figure **??**.

`exec('analog12.code')` Example of response of a filter obtained by `zpell`zpellps

4.2 Design of IIR Filters From Analog Filters

One way of designing IIR filters is by making discrete approximations to analog filters. If an approximation method is to be used then it is desirable to verify that important design characteristics of the analog filter are preserved by the approximation. A stable, causal analog filter design should yield a stable, causal digital filter under any approximation technique. Furthermore, essential characteristics of the analog filter should be preserved. For example, an analog low pass filter satisfies certain design characteristics such as the location of the cut-off frequency, the width of the transition band, and the amount of error in the pass and stop bands. The approximation of an analog filter should preserve these design specifications.

One approach to approximating an analog filter design is to sample the impulse response of the analog filter. This is known as the impulse invariance approximation. The relationship between the analog and discrete transfer functions under this approximation is

$$H(z)|_{z=e^{sT}} = \frac{1}{T}\sum_{k=-\infty}^{\infty} H(s + j\frac{2\pi k}{T}). \tag{4.12}$$

The approximation in (4.12) takes $z = e^{sT}$. Consequently, the left half s-plane maps into the unit circle in the z-plane, the right half s-plane maps outside the unit circle, and the $j\omega$-axis in the s-plane maps to the unit circle in the z-plane. Thus, this approximation technique preserves causal, stable filters. However, since (4.12) consists of a superposition of shifted versions of $H(s)$ along the $j\omega$-axis, aliasing can occur if the analog filter is not bandlimited.

Because most analog filter design techniques do not yield bandlimited filters aliasing is a problem. For example, a high pass filter cannot be bandlimited. Furthermore, because of aliasing distortion, filter specifications pertaining to band widths and errors are not necessarily preserved under the impulse invariance approximation.

In the following section two alternative approximation techniques are discussed. Each of these techniques avoids problems associated with aliasing.

4.3 Approximation of Analog Filters

4.3.1 Approximation of the Derivative

Consider an analog filter which can be represented by a rational transfer function, $H(s)$, where

$$H(s) = B(s)/A(s) \tag{4.13}$$

and $A(s)$ and $B(s)$ are polynomial functions of s. The relationship between the input, $X(s)$, and the output, $Y(s)$, of the filter in (4.13) can be expressed as

$$Y(s) = H(s)X(s) \tag{4.14}$$

or because of the rational nature of $H(s)$

$$[\sum_{n=0}^{N} a_n s^n]Y(s) = [\sum_{m=0}^{M} b_m s^m]X(s) \tag{4.15}$$

where the $\{a_n\}$ and the $\{b_m\}$ are the coefficients of the polynomial functions $A(s)$ and $B(s)$, respectively.

The relationship between the input and the output in the time domain is directly inferred from (4.15),

$$\sum_{n=0}^{N} a_n \frac{d^n}{dt^n} y(t) = \sum_{m=0}^{M} b_m \frac{d^m}{dt^m} x(t). \tag{4.16}$$

The differential equation in (4.16) can be approximated by using the Backward Difference Formula approximation to the derivative. That is, for T small we take

$$y'(t)|_{nT} \approx \frac{y(nT) - y(nT - T)}{T}. \tag{4.17}$$

Because the operation in (4.17) is linear and time-invariant the approximation can be represented by the z-transform,

$$Z\{y'(n)\} = (\frac{1 - z^{-1}}{T})Z\{y(n)\} \tag{4.18}$$

where $Z\{\cdot\}$ represents the z-transform operation and $y'(n)$ and $y(n)$ are sampled sequences of the time functions $y'(t)$ and $y(t)$, respectively.

Higher order derivatives can be approximated by repeated application of (4.17) which in turn can be represented with the z-transform by repeated multiplication by the factor $(1 - z^{-1})/T$. Consequently, the result in (4.16) can be approximately represented by the z-transform as

$$[\sum_{n=0}^{N} a_n(\frac{1-z^{-1}}{T})^n]Y(z) = [\sum_{m=0}^{M} b_m(\frac{1-z^{-1}}{T})^m]X(z). \tag{4.19}$$

Comparing (4.19) to (4.15) allows an identification of a transform from the s-plane to the z-plane,

$$s = \frac{1-z^{-1}}{T}. \tag{4.20}$$

Solving (4.20) for z yields

$$z = \frac{1}{1-sT}. \tag{4.21}$$

which can be rewritten and evaluated for $s = j\Omega$ as

$$z = \frac{1}{2}[1 + \frac{1+j\Omega T}{1-j\Omega T}]. \tag{4.22}$$

From (4.22) it can be seen that the $j\Omega$-axis in the s-plane maps to a circle of radius 1/2 centered at 1/2 on the real axis in the z-plane. The left half s-plane maps to the interior of this circle and the right half s-plane maps to the exterior of the circle. Figure **??** illustrates this transformation. `exec('iir1.code')` Transform $s = (1 - z^{-1})/T$f.iir1

The transform in (4.20) yields stable causal discrete filters when the analog filter is stable and causal. However, since the $j\Omega$-axis in the s-plane does not map to the unit circle in the z-plane it is clear that the frequency response of the digital filter will be a distorted version of the frequency response of the analog filter. This distortion may be acceptable in the case where the frequency response of the analog filter is bandlimited and the sampling period, T, is much higher than the Nyquist rate. Under these conditions the transformed frequency response is concentrated on the small circle in Figure **??** near the point $z = 1$ and the frequency response on the unit circle is less distorted.

4.3.2 Approximation of the Integral

An alternative approach to approximating the derivative of $y(t)$ is to approximate the integral of $y'(t)$. For example if $y(t)|_{nT-T}$ is known, then $y(t)|_{nT}$ can be approximated by the trapezoidal approximation rule

$$y(t)|_{nT} = \frac{T}{2}[y'(t)|_{nT} - y'(t)|_{nT-T}] + y(t)|_{nT-T}. \tag{4.23}$$

Taking the z-transform of the sequences $y'(n)$ and $y(n)$ yields the relationship

$$Z\{y'(n)\} = \frac{2}{T}[\frac{1-z^{-1}}{1+z^{-1}}]Z\{y(n)\} \tag{4.24}$$

and as before, we can substitute (4.24) into (4.16) and then make a correspondence with (4.15) yielding the transform

$$s = \frac{2}{T}[\frac{1-z^{-1}}{1+z^{-1}}] \tag{4.25}$$

between the s-plane and the z-plane. The expression in (4.25) is known as the bilinear transform.

Solving (4.25) for z yields

$$z = \frac{1 + (sT/2)}{1 - (sT/2)} \tag{4.26}$$

and evaluating (4.26) for $s = j\Omega$ gives

$$z = \frac{1 + (j\Omega T/2)}{1 - (j\Omega T/2)}. \tag{4.27}$$

The expression in (4.27) is an all-pass transformation which has unit magnitude and phase which takes values from $-\pi$ to π on the unit circle as Ω goes from $-\infty$ to ∞. The transformation in (4.26) maps the left half s-plane into the unit circle in the z-plane and the right half s-plane outside of the unit circle in the z-plane. Consequently, stable causal analog filters yield stable causal digital filters under this transformation. Furthermore, the $j\Omega$-axis in the s-plane maps to the unit circle in the z-plane. The mapping of the $j\Omega$-axis onto the unit circle is not linear and, thus, there is a frequency warping distortion of the analog filter design when this transform is used.

Because many filters of interest, such as low pass, band pass, band pass, and stop band filters have magnitudes which are piece-wise constant, frequency warping distortion is of no consequence. That is, the bilinear transformation maintains the characteristics of the analog filter design. However, if, in addition, the phase of the analog filter is linear, the bilinear transformation will destroy this property when used to obtain a digital filter.

4.4 Design of Low Pass Filters

For piece-wise constant specifications, the bilinear transform is the best of the three possible transforms discussed for converting analog filter designs into digital filter designs. Here we discuss how to use the bilinear transform to design a standard digital low pass filter. The next section presents a series of other transformations which can be used to convert a digital low pass filter into a high pass, band pass, stop band, or another low pass filter.

To effectively use the bilinear transform to design digital filters, it is necessary to transform the digital filter constraints into analog filter constraints. Evaluating (4.25) at $z = e^{j\omega}$ yields

$$s = \frac{2j}{T}\tan(\omega/2) = \sigma + j\Omega. \tag{4.28}$$

Thus, a digital filter constraint at ω corresponds to an analog filter constraint at

$$\Omega = \frac{2}{T}\tan(\omega/2). \tag{4.29}$$

Consequently, to design a digital low pass filter with cut-off frequency ω_c first requires an analog low pass filter design with cut-off frequency

$$\Omega_c = 2\tan(\omega_c/2). \tag{4.30}$$

(where we have used (4.29) with $T = 1$).

Any of the analog low pass filter design techniques already discussed (such as the Butterworth, Chebyshev, and elliptic filter designs) can be used to obtain the digital low pass filter design. The choice of model order can be made by specifying additional constraints on the filter design. For example, specification of a certain amount of attenuation at a specified frequency in the stop band can be used to obtain the model order of a Butterworth Filter. Such a specification for a digital filter would be converted to an analog filter specification using (4.29) before designing the analog filter. More on filter order specification can be found in the section on analog filter design.

An example of a typical digital low-pass filter design from a Chebyshev analog filter design of the first type is as follows. The digital low-pass filter is to have a cut-off frequency of $\pi/2$. This constraint is transformed to an analog constraint using (4.30). The resulting analog constraint takes the cut-off frequency to be $2\tan(\pi/4) = 2$. Now the function `zpch1` is used to design the Chebyshev filter of the first type of order 3 and passband ripple of .05. Since the ripple of a Chebyshev filter is $1/(1+\epsilon^2)$ it follows that for a ripple of .05 in the passband that $\epsilon = \sqrt{(1/.95)-1} = .22942$. Thus, the call to the function looks like

where the transfer function `hs` is calculated from the gain and the poles returned from the function. The magnitude of the the transfer function can be plotted as follows

which is displayed in Figure ??. `exec('iir2_3.code')` Magnitude of Analog Filterf.iir2

Now the analog low-pass filter design can be transformed to a digital filter design using the bilinear transform as follows

The result of the transform yields a filter which has a magnitude as shown in Figure ??. `exec('iir2_3.code')` Magnitude of Digital Filterf.iir3

4.5 Transforming Low Pass Filters

The previous section discussed the design of IIR low-pass filters based on the bilinear transform approximation to an analog low-pass filter. It is possible to transform a digital low-pass filter to a high-pass filter, band-pass filter, stop-band filter, or to another low-pass filter by using transforms similar to the bilinear transformation. This section presents the necessary transformations for each of the above filter types. The development follows [24].

Assuming that the cut-off frequency of the original digital low-pass filter is ω_c a new low-pass filter of cut-off frequency ω_u can be created using the following transformation

$$z \rightarrow \frac{z-\alpha}{1-z\alpha} \tag{4.31}$$

where

$$\alpha = \frac{\sin[(\omega_c-\omega_u)/2]}{\sin[(\omega_c+\omega_u)/2]}. \tag{4.32}$$

For a high-pass filter of cut-off frequency ω_u one can use the transformation

$$z \rightarrow -\frac{z+\alpha}{1+z\alpha} \tag{4.33}$$

where

$$\alpha = -\frac{\cos[(\omega_c-\omega_u)/2]}{\cos[(\omega_c+\omega_u)/2]}. \tag{4.34}$$

For a band-pass filter with ω_u and ω_l the upper and lower band edges, respectively, one would use the transformation

$$z \rightarrow -\frac{z^2-(2\alpha k/(k+1))z+((k-1)/(k+1))}{1-(2\alpha k/(k+1))z+((k-1)/(k+1))z^2} \tag{4.35}$$

where

$$\alpha = \frac{\cos[(\omega_u + \omega_l)/2]}{\cos[(\omega_u - \omega_l)/2]} \tag{4.36}$$

and

$$k = \cot[(\omega_u - \omega_l)/2]\tan(\omega_c/2). \tag{4.37}$$

Finally, for a stop-band filter with ω_u and ω_l the upper and lower band edges, respectivly, the following transformation is used

$$z \longrightarrow \frac{z^2 - (2\alpha/(k+1))z - ((k-1)/(k+1))}{1 - (2\alpha/(k+1))z - ((k-1)/(k+1))z^2} \tag{4.38}$$

where

$$\alpha = \frac{\cos[(\omega_u + \omega_l)/2]}{\cos[(\omega_u - \omega_l)/2]} \tag{4.39}$$

and

$$k = \tan[(\omega_u - \omega_l)/2]\tan(\omega_c/2). \tag{4.40}$$

4.6 How to Use the Function iir

The call to the function `iir` has the following syntax

```
--> [hz]=iir(n,ftype,fdesign,frq,delta)
```

The argument `n` is the filter order which must be a positive integer. The argument `ftype` is the filter type and can take values `'lp'` for a low-pass filter, `'hp'` for a high-pass filter, `'bp'` for a band-pass filter, or `'sb'` for a stop-band filter.

The argument `fdesign` indicates the type of analog filter design technique is to be used to design the filter. The value of `fdesign` can be `'butt'` for a Butterworth filter design, `'cheb1'` for a Chebyshev filter design of the first type, `'cheb2'` for a Chebyshev filter design of the second type, or `'ellip'` for an elliptic filter design.

The argument `frq` is a two-vector which contains cut-off frequencies of the desired filter. For low-pass and high-pass filters only the first element of this vector is used. The first element indicates the cut-off frequency of the desired filter. The second element of this vector is used for band-pass and stop-band filters. This second element is the upper band edge of the band-pass or stop-band filter, whereas the first element of the vector is the lower band edge.

The argument `delta` is a two-vector of ripple values. In the case of the Butterworth filter, `delta` is not used. For Chebyshev filters of the first type, only the first element of this vector is used and it serves as the value of the ripple in the pass band. Consequently, the magnitude of a Chebyshev filter of the first type ripples between 1 and 1-`delta(1)` in the pass band. For a Chebyshev filter of the second type only the second element of `delta` is used. This value of `delta` is the ripple in the stop band of the filter. Consequently, the magnitude of a Chebyshev filter of the second type ripples between 0 and `delta(2)` in the stop band. Finally, for the elliptic filter, both the values of the first and second elements of the vector `delta` are used and they are the ripple errors in the pass and stop bands, respectively.

The output of the function, `hz`, is a rational polynomial giving the coefficients of the desired filter.

4.7 Examples

In this section we present two examples using the `iir` filter design function. We remind the user that an important part of the filter design process is that there is always a trade-off between the performance and the expense of a filter design. For a filter with a small error in the pass and stop bands and with a narrow transition band it will be necessary to implement a filter of higher order (which requires more multiplies). Consequently, the filter design procedure is iterative. The user specifies a model order and then examines the magnitude of the resulting filter to see if the design specifications are met. If specifications are not satisfied, then the user must start again with a filter of higher model order. Another important point to keep in mind when using the function is that band pass and stop band filters will generate transfer functions of twice the model order specified. This is due to that transformation of the prototype low pass filter using an all pass filter of order two (see Section 4.5).

The first example is of a low-pass filter design using a Chebyshev filter of the first type for the analog design. The cut-off frequency of the digital filter is $\omega_c = .2$, the filter order is $n = 5$, and the ripple in the passband is $\delta = .05$. The call to the function is as follows

The result of the filter design is displayed in Figure **??** `exec('iir4.code')` Digital Low-Pass Filterf.iir4

The second example is of a band-pass filter designed from a third order analog elliptic filter with cut-frequencies $\omega_l = .15$ and $\omega_h = .25$ and ripples in the pass and stop bands, respectively, as $\delta_p = .08$ and $\delta_s = .03$. The call to Scilab looks like

and the resulting magnitude of the transfer function is illustrated in Figure **??**. `exec('iir5.code')` Digital Band-Pass Filterf.iir5 Notice that the transfer function here is of order six whereas the specified order was three. For band pass and stop band filters the user must specify a filter order of half the desired order to obtain the desired result.

4.8 Another Implementation of Digital IIR Filters

4.8.1 The `eqiir` function

The `eqiir` function is an interface between Scilab and the Fortran routine `syredi` which is a modification of the well known `eqiir` code [23]. The `eqiir` function allows one to design four different types of filters, namely lowpass, highpass, symmetric stopband, and symmetric passband filters. The algorithm is based on the bilinear transform of analog filters as described in the previous sections. The filter obtained is a product of second order cells. The order of the filter is computed automatically to meet the filter specifications.

The filter is given either by the set of its poles and zeros (output variables `zpoles` and `zzeros` of the `eqiir` function) or equivalently by a the representation:

$$H(z) = fact \prod_{1}^{N} n_i(z)/d_i(z)$$

where the rational fraction $n_i(z)/d_i(z)$ is the i-th element of `cells`.

4.8.2 Examples

Example 1 (Lowpass elliptic filter): Design of a lowpass elliptic filter with maximum ripples $\delta_p = 0.02$, $\delta_s = 0.001$ and cutoff frequencies $\omega_1 = 2\frac{\pi}{10}$ and $\omega_2 = 4\frac{\pi}{10}$.

Example 2 (Lowpass Butterworth filter): Design of a lowpass Butterworth filter with the following specifications:
- 5dB passband attenuation at the normalized cutoff frequencies $.25\pi/10$ and - 120dB attenuation at the stopband edge $0.5\pi/10$.

Example 3 (Lowpass type 1 Chebyshev filter): Design of a lowpass type 1 Chebyshev filter with 2dB ripple in the passband and -30 dB attenuation at the stopband edge. The sampling frequency is assumed to be 3000Hz and the cutoff frequencies at 37.5Hz and 75Hz respectively.

Example 4 (Elliptic symmetric bandpass filter): Design of a symmetric bandpass filter with edges at $\omega_1 = 0.251463, \omega_2 = \pi/10, \omega_3 = 2\pi/10, \omega_4 = 0.773302$ and ripples in the passband and stopband given respectively by $\delta_p = 0.022763$, $\delta_s = 0.01$.

The preceding example shows how to compute the magnitude response by using the `freq` primitive. The example is plotted in Figure **??**.

`exec('eqiir4.code')` Example of response obtained with `eqiir`ell4

Chapter 5

Spectral Estimation

5.1 Estimation of Power Spectra

The power spectrum of a deterministic, finite length, discrete-time signal, $x(n)$, is defined to be the magnitude squared of the signal's Fourier transform

$$S_x(\omega) = \frac{1}{N} | \sum_{n=0}^{N-1} x(n) e^{-j\omega n} |^2. \tag{5.1}$$

In an analogous fashion the cross-spectrum of two signals $x(n)$ and $y(n)$ is defined to be

$$S_{xy}(\omega) = \frac{1}{N} (\sum_{n=0}^{N-1} x(n) e^{-j\omega n}) (\sum_{n=0}^{N-1} y(n) e^{-j\omega n})^*. \tag{5.2}$$

The power spectra of random, zero-mean, wide sense stationary signals are obtained from the Fourier transform of the correlation functions of these signals. Thus, for R_x representing the autocorrelation function of x and R_{xy} representing the cross-correlation function of x with y we have by definition that

$$R_x(m) = E\{x(n+m)x^*(n)\} \tag{5.3}$$

$$R_{xy}(m) = E\{x(n+m)y^*(n)\}. \tag{5.4}$$

Consequently, the power spectrum and cross-power spectrum of $x(n)$ and of $x(n)$ with $y(n)$ are, respectively,

$$S_x(\omega) = \sum_{m=-\infty}^{\infty} R_x(m) e^{-j\omega m} \tag{5.5}$$

$$S_{xy}(\omega) = \sum_{m=-\infty}^{\infty} R_{xy}(m) e^{-j\omega m}. \tag{5.6}$$

The formulas (5.5) and (5.6) require estimates of the correlation functions. Possible candidates for the estimates of the auto and cross correlation functions of finite length random signals (i.e., $x(n) \neq 0$ and $y(n) \neq 0$ for $n = 0, 1, \ldots, N-1$) are

$$\hat{R}_x(m) = \frac{1}{N} \sum_{n=0}^{N-1-m} x(n+m)x^*(n) \tag{5.7}$$

$$\hat{R}_{xy}(m) = \frac{1}{N} \sum_{n=0}^{N-1-m} x(n+m)y^*(n). \tag{5.8}$$

The estimates in (5.7) and (5.8) are unbiased, consistent estimators in the limit as $N \rightarrow \infty$. Furthermore, in the case where the random signals are jointly Gaussian, these estimators are the maximum likelihood estimates of the correlation functions. Another interesting property of the estimators in (5.7) and (5.8) is that when substituted, respectively, into the expressions in (5.5) and (5.6), after some algebraic manipulation, yield exactly the expressions in (5.1) and (5.2).

Unfortunately, there is a serious problem with the above power spectrum estimation scheme. This problem is that the resulting power spectral estimates, in the limit, are not consistent. That is, the error variance of the estimate does not decrease with increasing data. Consequently, power spectral estimates obtained by taking the magnitude squared of the Fourier transform are high-variance, low-quality estimates.

In the sections which follow two techniques are discussed which yield improved spectral estimates. These techniques are both based on averaging spectral estimates obtained from the classical approach just described. This averaging, although introducing some biasing, yields greatly improved estimates in that, in the limit, these estimates become consistent.

The first averaging technique also sections the data into overlapping segments. However, in this case the magnitude squared of the Fourier transform is calculated from each segment and then these are averaged together to yield the spectral estimate. This technique is called the modified periodogram method for spectral estimation.

The second averaging technique sections the data into overlapping segments. For each segment an estimate of the correlation function is calculated. These estimates are then averaged and the estimated power spectral density is the Fourier transform of the average. This technique is known as the correlation method for spectral estimation.

Both techniques use windows to diminish the effects that finite data has on spectral estimation. These effects are identical to the problems encountered in FIR filter design, and, consequently, the reader is referred to the FIR filter design section for an explanation of the issues involved in the choice of windows. In the discussion which follows cross-spectral estimation is not discussed considering that the cross-spectral estimate can be obtained as a simple modification of the auto-spectral estimation techniques.

5.2 The Modified Periodogram Method

The modified periodogram method of spectral estimation repeatedly calculates the periodogram of windowed sub-sections of the data. These periodograms are then averaged together and normalized by an appropriate constant to obtain the final spectral estimate. It is the averaging process which reduces the variance in the estimate.

The periodogram of a finite data sequence is defined by

$$I(\omega) = \frac{1}{U} | \sum_{n=0}^{N-1} w(n)x(n)e^{-j\omega n}|^2. \tag{5.9}$$

where $w(n)$ is a window function which has energy U. Consequently, if K sub-segments of length N are used to calculate the spectrum of the signal then the modified periodogram spectral estimate, $\hat{S}_x$, is just the average of the K periodograms

$$\hat{S}_x(\omega) = \frac{1}{K} \sum_{k=0}^{K-1} I_k \tag{5.10}$$

where each of the I_k is the periodogram (calculated as in (5.9)) of the k^{th} segment of data.

Normally, the K segments of data are taken so that there is a regular pattern of overlap in the successive segments. That is, the k^{th} and $(k+1)^{th}$ segments overlap by D points. Figure **??** illustrates three consecutive overlapping segments of a data sequence. `exec('spect1.code')` Overlapping Dataf.spect1 It can be shown that an overlap of fifty percent in the data segments results in an approximately fifty percent reduction in the variance of the estimate of the power spectrum [24]. Normally, one chooses the length of the data segments to reflect the *a priori* knowledge of the correlation length of the data. That is to say that if the correlation between two data samples separated by more than M points is considered negligible then the data segment should be of a length on the order of M. The number of data segments used determines the variance of the spectral estimate. The variance decreases proportionally to the number of independent segments. Of course, with a limited quantity of data the number of data segments must also be limited.

The function `pspect` calculates an estimate of the power spectrum using the modified periodogram method.

5.2.1 Example Using the `pspect` function

In this section, we demonstrate the use of the `pspect` macro. The data used is generated by passing zero-mean white noise of unit variance through a low-pass filter. Consequently, the spectrum of the data should be the magnitude square of the filter frequency response. The low-pass filter is an FIR filter of length 33 generated using the function `eqfir` .

The data was generated using the following Scilab commands,

As can be seen, a total of 1024 data points are available for the estimation of the spectrum. The logarithm of the magnitude squared of the filter frequency response is shown in Figure **??**. `exec('spect2_4.code')` Log Magnitude Squared of Filterf.spect2

The data obtained above are used to estimate the power spectrum in the following way

```
-->[sm]=pspect(100,200,'tr',y);
```

The log-magnitude of the power spectrum (`sm`) is plotted in Figure **??**. It should be pointed out here that the value of the section lengths was chosen to be 200 in this example to obtain additional resolution in the plot of the Fourier transform of the estimated power spectrum in Figure **??**. However, there is very acceptable behavior of the spectral estimate when the section length is on the order of twice the filter length. This is due to the fact that one does not expect correlations in the data that are much longer than the filter length. Normally, the section lengths are chosen to reflect the *a priori* knowledge of the correlation length in the data. `exec('spect2_4.code')` Estimate of Spectrumf.spect3 As can be seen the estimated spectrum matches the theoretical spectrum (Figure **??**) very well. In particular, the peaks of the spectrum in both the pass and stop bands matches well with those of the filter magnitude response. Furthermore, the normalization of the estimate is accurate with respect to the filter response.

5.3 The Correlation Method

The correlation method for power spectral estimation calculates the spectral estimate as the Fourier transform of a modified estimate of the autocorrelation function. This modified estimate of the autocorrelation function consists of repeatedly calculating estimates of the autocorrelation function as in (5.7) from overlapping sub-segments of the data, and then averaging these estimates to obtain the modified estimate.

Consequently, referring again to Figure ??, for each length N sub-segment of the data $x_k(n)$ the estimate of $2M$ points of the autocorrelation function is calculated by

$$\hat{R}_k(m) = \sum_{n=0}^{N-1-m} x(n+m)x^*(n) \tag{5.11}$$

for $m = 0, \pm 1, \pm 2, \ldots, \pm M$. For K estimates of the autocorrelation function calculated as in (5.11) the power spectral estimate is obtained from

$$\hat{S}_x(\omega) = \mathcal{F}\{\tilde{R}_x(m)w(m)\} \tag{5.12}$$

where $\mathcal{F}\{\cdot\}$ represents the Fourier transform operation, $w(m)$ is a window function, and $\tilde{R}_x(m)$ is the average of the K estimates

$$\tilde{R}_x = \frac{1}{K}\sum_{k=1}^{K} \hat{R}_k. \tag{5.13}$$

The correlation method of spectral estimation is based on the `corr` primitive in Scilab. The primitive `corr` is useful for any application requiring correlations or cross-correlations. Documentation on this primitive can be found in the introductory manual for Scilab.

The function `cspect` calculates an estimate of the power spectrum using the correlation method for spectral estimation.

5.3.1 Example Using the function `cspect`

Here, for comparison purposes, the same example as used in the case of the `pspect` macro is examined using the `cspect` macro. The data used is identical to that used in the previous example. These data are used to estimate the power spectrum in the following way

```
-->[sm]=cspect(100,200,'tr',y);
```

The log-magnitude of the power spectrum (`sm`) is plotted in Figure ??.

It should be pointed out here that the value of the the number of lags (100) and the number of transform points (200) were chosen to match the previous example where the `pspect` macro was used. A plot of the estimated power spectrum is illustrated in Figure ??. `exec('spect2_4.code')` Estimate of Spectrumf.spect4 As can be seen the estimated spectrum also matches the theoretical spectrum (Figure ?? very well. There are some differences, however, between the estimates obtained using the two different macros. The primary difference to keep in mind is the difference in how the windows are used for the two different techniques. In the correlation method the magnitude of the window is convolved with the spectrum of the signal. In the modified periodogram method the *square* of the magnitude of the window is convolved with the spectrum of the signal. Consequently, the effects of windows are different in each of the two cases (for example, the side-lobes of the window are lower in the case of the modified periodogram method due to the squaring of its magnitude). The quantitative differences between the two techniques are difficult to address here due to the complexity of the question. There are some relevant questions concerning which technique may be the best in any one application. For more information on how to choose between the techniques the user is referred to [24] and the relevant references found within.

5.4 The Maximum Entropy Method

5.4.1 Introduction

The power spectrum of a deterministic signal is defined to be the squared magnitude of the signal's Fourier transform. That is, for $x(n)$ a discrete time signal, the power spectrum, $S_x(\omega)$, is

$$S_x(\omega) = |X(\omega)|^2 \tag{5.14}$$

where

$$X(\omega) = \sum_{n=-\infty}^{\infty} x_n e^{-j\omega n}. \tag{5.15}$$

In many applications it is very useful to know the power spectrum of a signal, however, it is rare that the obtained signal can be characterized as being deterministic. Often the signal is present in a noisy environment and, in addition, is obtained with instrumentation which degrades the signal with measurement noise. Consequently, for a non-deterministic signal one seeks to estimate the power spectrum. It can be shown [21] that taking the Fourier transform of a non-deterministic signal in order to estimate its power spectrum is a very poor approach. The problem is that the resulting power spectrum is a highly variable estimate of the true power spectrum and that the variance does not decrease to zero as one increases the data length (i.e., the estimator is not consistent).

The problem of estimating the power spectrum can be modified as follows. Let $x(n)$ be a zero-mean, stationary signal and let $r_x(n)$ be the autocorrelation function of the signal (that is, $r_x(n) = E\{x(k)x^*(n+k)\}$). Then the power spectrum $S_x(n)$ of $x(n)$ is taken to be the Fourier transform of $r_x(n)$

$$S_x(\omega) = \sum_{n=-\infty}^{\infty} r_x(n) e^{-j\omega n}. \tag{5.16}$$

Assuming that statistical averages of the signal $x(n)$ are equal to time averages we can take as an estimate for $r_x(n)$

$$\hat{r}_x(n) = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{m=-N}^{N} x(m)x(m-n). \tag{5.17}$$

However, after plugging (5.17) into (5.16) and performing some algebraic manipulation it can be seen that (5.16) is just the magnitude squared of the Fourier transform of $x(n)$. Consequently, (5.16) is not any more useful than (5.14) for estimating the power spectrum of a non-deterministic signal.

One can improve the statistical properties of the power spectrum estimate by smoothing the result obtained in (5.16) or by breaking the input, $x(n)$, into pieces, performing the calculation in (5.17) and (5.16) for each piece, and then averaging these results together. These approaches are the classical methods of power spectral estimation.

These classical methods of power spectral estimation are undesirable for two reasons. The estimate obtained from these methods is based on a finite (i.e., windowed) segment of the autocorrelation function. This means that the resulting estimated power spectrum is a convolution of the true power spectrum and of the Fourier transform of the window. Consequently, the resolution of spectral features is diminished and the estimate of the power spectrum at any point is biased by leakage from other parts of the power spectrum through the window sidelobes.

The maximum entropy spectral estimate (MESE) of the power spectrum yields an improved spectral density estimate. That's to say that for MESE the resolution is improved and the bias is decreased in comparison with the classical spectral estimation methods. This improvement is due to the fact that the MESE uses a model based estimation procedure.

5.4.2 The Maximum Entropy Spectral Estimate

The maximum entropy spectral estimate (MESE) is designed to produce high-resolution, low-bias spectral estimates from finite length discrete signals. The formulation of the MESE problem is as follows. It is assumed that only a finite number, N, of autocorrelation lags (estimated from a finite length discrete signal) are available. The MESE yields the function $\hat{S}_x(\omega)$ which has maximum entropy and whose inverse Fourier transform exactly matches the N lags, $\hat{r}_x(n)$. This can be expressed by the equation

$$\hat{S}_x(\omega) = \max_{S(\omega)}\{-\int_{-\pi}^{\pi} S(\omega)\log[S(\omega)]d\omega\} \tag{5.18}$$

where

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \hat{S}_x(\omega)e^{j\omega n}d\omega = \hat{r}_x(n), \quad n = 0, 1, \ldots, N-1. \tag{5.19}$$

Equation (5.18) expresses the optimality condition of maximizing the entropy of $S(\omega)$ subject to the N constraints posed in (5.19).

Since entropy is a measure of randomness, the MESE is the spectral estimate which is maximally random given the constraints in (5.19). Intuitively, the MESE incorporates no information in the estimated spectrum other than the knowledge of the autocorrelation lags. That is to say that the bias should be eliminated (or at least minimized in some sense) since no non-data related constraints are imposed on the spectrum. As was discussed in the introduction, windowed-autocorrelation spectral estimates suffered from bias due to the leakage from the window sidelobes. The window imposes a non-data related constraint on the power spectrum estimate in that the autocorrelation function is assumed to be identically zero outside of the support of the window.

Furthermore, as is discussed in [16], it can be shown that the MESE is equivalent to the Fourier transform of an infinite length autocorrelation sequence which is obtained by extrapolating the sequence of length N in (5.19). The extrapolation is accomplished using an auto-regressive, all-pole model of order $N-1$ given by

$$\hat{r}_x(n) = -\sum_{k=1}^{N-1} a_k\hat{r}_{n-k}, \quad n \geq N. \tag{5.20}$$

Any autocorrelation sequence can be modeled by (5.20) given a large enough model order, N. Consequently, in principle, the resolution of the MESE should be much better than that of a windowed spectral estimate since the MESE uses an infinite length autocorrelation sequence.

The solution to (5.18) and (5.19) can be found by using the calculus of variations [11] and, as demonstrated in [16], the solution takes the form

$$\hat{S}_x(\omega) = \frac{\sigma^2}{|1 + \sum_{n=1}^{N-1} a_n \exp\{-j\omega n\}|^2} \tag{5.21}$$

where the parameter set $\{\sigma^2, a_1, a_2, \ldots, a_{N-1}\}$ is obtained by solving the system of linear equations

$$\begin{bmatrix} \hat{r}_x(0) & \hat{r}_x(1) & \cdots & \hat{r}_x(N-1) \\ \hat{r}_x(1) & \hat{r}_x(0) & \cdots & \hat{r}_x(N-2) \\ \vdots & \vdots & & \vdots \\ \hat{r}_x(N-1) & \hat{r}_x(N-2) & \cdots & \hat{r}_x(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.22}$$

where the Toeplitz matrix in (5.22) is composed of the N estimated correlation lags $\hat{r}_x(n)$. The system of N linear equations in (5.22) are known as the Yule-Walker equations and an efficient algorithm for their solution is described in the next section.

5.4.3 The Levinson Algorithm

An efficient recursive solution to the Yule-Walker equations in (5.22) exists and is known as the Levinson algorithm. The algorithm requires $O(N^2)$ complex multiplications and additions. The solution to the k^{th} order problem is obtained from the solution to the $(k-1)^{th}$ order problem using the following equations

$$\begin{aligned} a_{kk} &= -[\hat{r}_x(k)+\sum_{j=1}^{k-1} a_{k-1,j}\hat{r}_x(k-j)]/\sigma^2_{k-1} && (5.23)\\ a_{ki} &= a_{k-1,i}+a_{kk}a^*_{k-1,k-i} && (5.24)\\ \sigma^2_k &= (1-|a_{kk}|^2)\sigma^2_{k-1}. && (5.25)\end{aligned}$$

The solution to the 1^{st} order problem is

$$\begin{aligned} a_{11} &= -\hat{r}_x(1)/\hat{r}_x(0) && (5.26)\\ \sigma^2_1 &= (1-|a_{11}|^2)\hat{r}_x(0). && (5.27)\end{aligned}$$

5.4.4 How to Use `mese`

The syntax for the macro `mese` is as follows,

```
-->[sm,fr]=mese(x)
```

where one wants to obtain a power spectral estimate of `x`, the input data sequence, and `sm` is the resulting estimate obtained on the normalized frequency axis ($0 \leq$`fr`$\leq .5$).

5.4.5 How to Use `lev`

The syntax for the macro `lev` is as follows,

```
-->[ar,sigma2,rc]=lev(r)
```

where `r` is a vector of auto-correlation coefficients $(r(0), r(1), \ldots, r(N-1))$, `ar` is the vector which satisfies the Yule-Walker equations, `sigma2` is the scalar which satisfies the Yule-Walker equations, and `rc` is a vector of reflection coefficients.

5.4.6 Examples

Here we give an example of estimating the power spectrum of a very short data sequence using the MESE and also using the magnitude squared of the Fourier transform. The data is eleven samples of the sum of two sinusoids in additive, uniformly distributed, white noise. The functional form of the data sequence is

$$x(n)=\sin(2\pi n/20)+\sin(3.5\pi n/20)+.2w(n) \qquad (5.28)$$

where $w(n)$ is the white noise sequence which takes on values in $[-1,1]$ and $n=0,1,...,10$. Figure **??** shows the input data sequence, x(n). Figures **??** and **??** show the maximum entropy and magnitude squared estimates of the power spectrum, respectively. `exec('mem1_3.code')` Input

Data Sequence, $x(n)$f33.1 `exec('mem1_3.code')` Maximum Entropy Spectral Estimate of $x(n)$f3.2 `exec('mem1_3.code')` Squared Magnitude of the Fourier Transform of $x(n)$f3.3 As can be seen, the MESE resolves two peaks according to the two sinusoidal frequences in $x(n)$. The squared magnitude of the Fourier transform of $x(n)$ does not have a long enough signal to resolve the two peaks of the spectrum. Furthermore, the power spectrum estimate in Figure **??** shows spurious sidelobes which have nothing to do with the data.

Chapter 6

Optimal Filtering and Smoothing

6.1 The Kalman Filter

Consider the following discrete-time system,

$$\begin{aligned} x_{k+1} &= F_k x_k + G_k w_k \\ y_k &= H_k x_k + v_k \end{aligned} \tag{6.1}$$

where

$$\begin{aligned} w_k &\sim N(0, Q_k) \\ v_k &\sim N(0, R_k) \\ x_0 &\sim N(m_0, \Pi_0) \end{aligned} \tag{6.2}$$

and x_0, $\{w_k\}$, and $\{v_k\}$ are independent random vectors. The state vector, x_k, has dimension N and the observation vector, y_k, has dimension M. Furthermore, it is assumed that $R_k > 0$.

The problem to be addressed here is the estimation of the state vector, x_k, given observations of the vectors $Y_k = \{y_0, y_1, \ldots, y_k\}$. Because the collection of variables $\{x_k, y_0, y_1, \ldots, y_k\}$ are jointly Gaussian one could estimate x_k by maximizing the likelihood of the conditional probability distribution $p(x_k|Y_k)$ given the values of the conditional variables. Equivalently, one could search the estimate, $\hat{x}_k$, which minimized the mean square error, $\epsilon_k = x_k - \hat{x}_k$. In either case it is known that the optimal estimate (maximum likelihood or least squares) for the jointly Gaussian variables is the conditional mean. The error in the estimate is the conditional covariance.

In what follows, the conditional mean and covariance of x_k given Y_k is developed. This is followed by a description of the Kalman filter, an extremely practical recursive method for calculating the conditional mean and covariance. Several different implementations of the Kalman filter are discussed here: The steady-state Kalman filter which can be used when the system matrices in (6.1) and (6.3) are non-time varying, the non-stationary Kalman filter for use when the system matrices in (6.1) and (6.3) are time-varying, and, finally, the square-root Kalman filter which is used (for time-varying and non-time-varying system matrices) when greater numerical accuracy is required.

6.1.1 Conditional Statistics of a Gaussian Random Vector

The minimum mean square estimate of a Gaussian random vector given observations of some of its elements is the conditional mean of the remaining elements. The error covariance of this estimate is the conditional covariance. Consequently, assuming that z is a Gaussian random vector composed of two sub-vectors x and y, then

$$z = \begin{bmatrix} x \\ y \end{bmatrix} \sim N\left(\begin{bmatrix} m_x \\ m_y \end{bmatrix}, \begin{bmatrix} \Lambda_x & \Lambda_{xy} \\ \Lambda_{yx} & \Lambda_y \end{bmatrix} \right) \tag{6.3}$$

where m_x denotes the mean of x, Λ_{xy} denotes the covariance of x with y, and Λ_x is the covariance of x with itself.

It is known that the marginal and conditional distributions of a Gaussian random vector are also Gaussian. In particular, the conditional distribution of x given y, $p(x|y)$, is

$$p(x|y) = N(m_{x|y}, \Lambda_{x|y}) \tag{6.4}$$

where $m_{x|y}$ denotes the conditional mean and $\Lambda_{x|y}$ denotes the conditional covariance. These two quantities may be calculated by

$$\begin{aligned} m_{x|y} &= m_x + \Lambda_{xy}\Lambda_y^{-1}(y - m_y) \\ \Lambda_{x|y} &= \Lambda_x - \Lambda_{xy}\Lambda_y^{-1}\Lambda_{yx}. \end{aligned} \tag{6.5}$$

(It is useful to note the x and y are not necessarily of the same dimension). Equation (6.5) is very important in our development of the Kalman filter equations.

With regard to the problem posed in the introduction to this section, the minimum mean square error is calculated in a straight forward manner. One stacks the individual observation vectors into a single vector, Y_k, and then, since x_k and Y_k are jointly Gaussian , one applies (6.5) to obtain the conditional mean and covariance of x_k given Y_k. The problem with this approach is that for increasing k the vector Y_k is of increasing dimension, and consequently, the matrix inverse and multiplication operations in (6.5) become increasingly burdensome.

The next few sections are devoted to showing how the linear system of (6.1) and the special properties of (6.5) can be used to obtain a recursive update to the estimation of x_k. That is, given the best estimate of x_k based on the observations Y_k (we denote this estimate by $\hat{x}_{k|k}$) and a new observation y_{k+1}, it is shown how to obtain the best estimate of $\hat{x}_{k+1|k+1}$ and its error covariance matrix $P_{k+1|k+1}$.

6.1.2 Linear Systems and Gaussian Random Vectors

Given a random vector, x, with a probability distribution $p(x)$, the minimum mean square error estimate of x is denoted here by $\hat{x}$ and consists of the mean of x. That is, $\hat{x} = m_x$. The associated error covariance of the estimate, denoted P_x, is the covariance of x. Now assume that x is passed through a linear system and is disturbed by an independent, zero-mean Gaussian vector, v, of covariance R. This yields an output which can be represented by

$$y = Hx + v. \tag{6.6}$$

Since (6.6) is a linear combination of independent Gaussian random vectors, y is also a Gaussian random vector. The mean and covariance of y are calculated as follows,

$$\begin{aligned} m_y &= E\{y\} \\ &= E\{Hx + v\} \\ &= Hm_x \end{aligned} \tag{6.7}$$

and

$$\begin{aligned}\Lambda_y &= E\{(y-m_y)(y-m_y)^T\}\\ &= E\{[H(x-m_x)+v][H(x-m_x)+v]^T\}\\ &= H\Lambda_x H^T + R. \qquad (6.8)\end{aligned}$$

Consequently, the minimum mean square error estimate of y is $\hat{y} = Hm_x$ and the associated error covariance of this estimate is $P_y = H\Lambda_x H^T + R = HP_x H^T + R$.

6.1.3 Recursive Estimation of Gaussian Random Vectors

Now we assume that we have a Gaussian random vector which is composed of three sub-vectors x, y, and z. This is represented by

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \sim N\left(\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix}, \begin{bmatrix} \Lambda_x & \Lambda_{xy} & \Lambda_{xz} \\ \Lambda_{yx} & \Lambda_y & \Lambda_{yz} \\ \Lambda_{zx} & \Lambda_{zy} & \Lambda_z \end{bmatrix}\right). \qquad (6.9)$$

From (6.5) the minimum mean square error estimate of x given observation of y is

$$\hat{x}(y) = m_x + \Lambda_{xy}\Lambda_y^{-1}(y-m_y) \qquad (6.10)$$

and the associated error covariance is

$$P_x(y) = \Lambda_x - \Lambda_{xy}\Lambda_y^{-1}\Lambda_{yx}. \qquad (6.11)$$

It is valuable to note here that

$$E\{\hat{x}(y)\} = m_x. \qquad (6.12)$$

If z is also observed, then the minimum mean squared error estimate of x given y and z is

$$\hat{x}(y,z) = m_x + \begin{bmatrix} \Lambda_{xy} & \Lambda_{xz} \end{bmatrix} \begin{bmatrix} \Lambda_y & \Lambda_{yz} \\ \Lambda_{zy} & \Lambda_z \end{bmatrix}^{-1} \begin{bmatrix} y-m_y \\ z-m_z \end{bmatrix} \qquad (6.13)$$

with error covariance

$$P_x(y,z) = \Lambda_x - \begin{bmatrix} \Lambda_{xy} & \Lambda_{xz} \end{bmatrix} \begin{bmatrix} \Lambda_y & \Lambda_{yz} \\ \Lambda_{zy} & \Lambda_z \end{bmatrix}^{-1} \begin{bmatrix} \Lambda_{yx} \\ \Lambda_{zx} \end{bmatrix}. \qquad (6.14)$$

Now note that if y and z were independent that Λ_{yz} would be zero and then (6.13) could be written as

$$\begin{aligned}\hat{x}(y,z) &= m_x + \Lambda_{xy}\Lambda_y^{-1}(y-m_y) + \Lambda_{xz}\Lambda_z^{-1}(z-m_z)\\ &= \hat{x}(y) + \Lambda_{xz}\Lambda_z^{-1}(z-m_z). \qquad (6.15)\end{aligned}$$

The result in (6.15) is a recursive method for calculating $\hat{x}(y,z)$ given $\hat{x}(y)$ and z. The problem is that (6.15) depends on y and z being independent random vectors. Fortunately, by a change of variables, one can always change the estimation procedure in (6.13) into that in (6.15). This is accomplished as follows. Let ν be a random vector defined by

$$\begin{aligned}\nu &= z - \hat{z}(y)\\ &= z - [m_z + \Lambda_{zy}\Lambda_y^{-1}(y-m_y)]\\ &= (z-m_z) - \Lambda_{zy}\Lambda_y^{-1}(y-m_y) \qquad (6.16)\end{aligned}$$

where $\hat{z}(y)$ is the minimum mean square estimate of z given observation of y (obtained by using (6.5)).

The new random vector, ν, has several interesting properties. First, because

$$m_\nu = E\{(z - m_z) - \Lambda_{zy}\Lambda_y^{-1}(y - m_y)\} = 0 \tag{6.17}$$

ν is a zero-mean random variable. Also, since

$$\begin{aligned}
\Lambda_{\nu y} &= E\{\nu(y - m_y)^T\} \\
&= E\{(z - m_z)(y - m_y)^T - \Lambda_{zy}\Lambda_y^{-1}(y - m_y)(y - m_y)^T\} \\
&= \Lambda_{zy} - \Lambda_{zy}\Lambda_y^{-1}\Lambda_y \\
&= 0
\end{aligned} \tag{6.18}$$

and

$$\begin{aligned}
\Lambda_{\nu\hat{x}(y)} &= E\{\nu(m_x + \Lambda_{xy}\Lambda_y^{-1}(y - m_y) - m_x)^T\} \\
&= E\{\nu(y - m_y)^T\Lambda_y^{-1}\Lambda_{yx}\} \\
&= 0
\end{aligned} \tag{6.19}$$

it follows that ν is independent of both y and $\hat{x}(y)$ These properties are very useful for developing the Kalman filter equations of the next section.

Now (6.15) can be rewritten so that

$$\begin{aligned}
\hat{x}(y, z) &= \hat{x}(y, \nu) \\
&= m_x + \left[\begin{array}{cc} \Lambda_{xy} & \Lambda_{x\nu} \end{array}\right] \left[\begin{array}{cc} \Lambda_y & 0 \\ 0 & \Lambda_\nu \end{array}\right]^{-1} \left[\begin{array}{c} y - m_y \\ \nu \end{array}\right] \\
&= m_x + \Lambda_{xy}\Lambda_y^{-1}(y - m_y) + \Lambda_{x\nu}\Lambda_\nu^{-1}\nu \\
&= \hat{x}(y) + \Lambda_{x\nu}\Lambda_\nu^{-1}\nu
\end{aligned} \tag{6.20}$$

where, from (6.16) we obtain

$$\begin{aligned}
\Lambda_{x\nu} &= E\{(x - m_x)(z - m_z - \Lambda_{zy}\Lambda_y^{-1}(y - m_y))^T\} \\
&= \Lambda_{xz} - \Lambda_{xy}\Lambda_y^{-1}\Lambda_{yz}
\end{aligned} \tag{6.21}$$

and

$$\begin{aligned}
\Lambda_\nu &= E\{(z - m_z - \Lambda_{zy}\Lambda_y^{-1}(y - m_y))(z - m_z - \Lambda_{zy}\Lambda_y^{-1}(y - m_y))^T\} \\
&= \Lambda_z - \Lambda_{zy}\Lambda_y^{-1}\Lambda_{yz}.
\end{aligned} \tag{6.22}$$

(Note that the equality of $\hat{x}(y, z)$ and $\hat{x}(y, \nu)$ is due to the fact that no information is lost in making the change of variables in (6.16). We are simply adding a constant vector to z which renders ν and y independent of each other). The error covariance, $P_x(y, \nu)$, associated with (6.20) is

$$\begin{aligned}
P_x(y, \nu) &= \Lambda_x - \left[\begin{array}{cc} \Lambda_{xy} & \Lambda_{x\nu} \end{array}\right] \left[\begin{array}{cc} \Lambda_y & 0 \\ 0 & \Lambda_\nu \end{array}\right]^{-1} \left[\begin{array}{c} \Lambda_{yx} \\ \Lambda_{\nu x} \end{array}\right] \\
&= \Lambda_x - \Lambda_{xy}\Lambda_y^{-1}\Lambda_{yx} - \Lambda_{x\nu}\Lambda_\nu^{-1}\Lambda_{\nu x} \\
&= P_x(y) - \Lambda_{x\nu}\Lambda_\nu^{-1}\Lambda_{\nu x}.
\end{aligned} \tag{6.23}$$

6.1.4 The Kalman Filter Equations

Here the results of the previous two sections are combined to find the recursive estimation procedure referred to in the introduction. Before detailing the procedure we introduce some notation. We denote by $\hat{x}_{k|l}$ the minimum mean square estimate of x_k given the observations $Y_l = \{y_0, y_1, \ldots, y_l\}$. Furthermore, $P_{k|l}$ represents the error covariance associated with $\hat{x}_{k|l}$.

Now, the estimate $\hat{x}_{k|k}$ can be obtained from the estimate $\hat{x}_{k|k-1}$ and the new observation y_k in the following manner. From (6.20) and (6.23) we have

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + \Lambda_{x_k \nu_k} \Lambda_{\nu_k}^{-1} \nu_k \\ P_{k|k} &= P_{k|k-1} - \Lambda_{x_k \nu_k} \Lambda_{\nu_k}^{-1} \Lambda_{\nu_k x_k} \end{aligned} \tag{6.24}$$

where from (6.1), (6.7), and (6.8)

$$\nu_k = y_k - H_k \hat{x}_{k|k-1}. \tag{6.25}$$

The covariance matrices in (6.24) may be calculated by

$$\begin{aligned} \Lambda_{\nu_k} &= E\{\nu_k \nu_k^T\} \\ &= E\{(y_k - H_k \hat{x}_{k|k-1})(y_k - H_k \hat{x}_{k|k-1})^T\} \\ &= E\{[H_k(x_k - \hat{x}_{k|k-1}) + v_k][H_k(x_k - \hat{x}_{k|k-1}) + v_k]^T\} \\ &= H_k P_{k|k-1} H_k^T + R_k \end{aligned} \tag{6.26}$$

(where the final equality follows from the fact that v_k and x_k are independent random vectors), and

$$\begin{aligned} \Lambda_{x_k \nu_k} &= E\{(x_k - E\{x_k\})\nu_k^T\} \\ &= E\{(x_k - E\{x_k\} + E\{x_k\} - \hat{x}_{k|k-1})\nu_k^T\} \\ &= E\{(x_k - \hat{x}_{k|k-1})\nu_k^T\} \\ &= E\{(x_k - \hat{x}_{k|k-1})(y_k - H_k \hat{x}_{k|k-1})^T\} \\ &= E\{(x_k - \hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})^T H_k^T\} \\ &= P_{k|k-1} H_k^T \end{aligned} \tag{6.27}$$

(where the second equality follows from (6.13) and (6.19)). Substituting (6.25), (6.26), and (6.27) into (6.24) yields

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - H_k \hat{x}_{k|k-1}) \\ P_{k|k} &= P_{k|k-1} - K_k H_k P_{k|k-1} \end{aligned} \tag{6.28}$$

where $K_k = P_{k|k-1} H_k^T [H_k P_{k|k-1} H_k^T + R_k]^{-1}$ is called the Kalman gain of the filter. (Note: Since the model proposed in the first section of this chapter assumes that $R_k > 0$ it follows that K_k always exists. However, when R_k is not strictly positive definite there may be problems performing the matrix inverse necessary to calculate K_k). Using (6.7) and (6.8) in conjunction with (6.1) gives the two auxiliary equations

$$\begin{aligned} \hat{x}_{k+1|k} &= F_k \hat{x}_{k|k} \\ P_{k+1|k} &= F_k P_{k|k} F_k^T + G_k Q_k G_k^T. \end{aligned} \tag{6.29}$$

Combining the equations in (6.28) and (6.29) yields one set of recursive equations

$$\begin{aligned} \hat{x}_{k+1|k} &= F_k \hat{x}_{k|k-1} + F_k K_k (y_k - H_k \hat{x}_{k|k-1}) \\ P_{k+1|k} &= F_k P_{k|k-1} F_k^T - F_k K_k H_k P_{k|k-1} F_k^T + G_k Q_k G_k^T. \end{aligned} \tag{6.30}$$

The only remaining detail is to identify the initial conditions of the Kalman filter which are obtained from the *a priori* statistics of x_0

$$\begin{aligned} \hat{x}_{0|-1} &= m_0 \\ P_{0|-1} &= \Pi_0. \end{aligned} \tag{6.31}$$

6.1.5 Asymptotic Properties of the Kalman Filter

Depending on the problem formulation posed in (6.1) there are situations where the Kalman filter does not perform well. That is to say, that for certain formulations, the Kalman filter could provide state estimates which diverge from the actual evolution of the state vector. This divergence is by no means a fault of the Kalman filter, but, rather, is due to the process model provided by the user. Consequently, in such a case where the Kalman filter diverges, the user may wish to re-examine the model formulation so that it is better posed for the estimation problem. The following results concerning the effectiveness of the Kalman filter are only valid for time invariant formulations of the model in (6.1). That is to say that $F_k = F$, $G_k = G$, $H_k = H$, $Q_k = Q$, and $R_k = R$ for all k.

We state the following properties of the asymptotic behavior of the Kalman filter without proof. The desirable asymptotic properties of the Kalman filter depend on the controllability and observability of the system model in (6.1). A necessary and sufficient condition for the system in (6.1) to be controllable is that

$$rank[\ G \quad FG \quad \cdots \quad F^{N-1}G\ ] = N \tag{6.32}$$

(recall that F is an $N \times N$ matrix). A necessary and sufficient condition that the system be observable is that

$$rank \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{N-1} \end{bmatrix} = N \tag{6.33}$$

Now if the system in (6.1) is both controllable and observable then

$$\lim_{k\to\infty} P_{k|k-1} = P < \infty \tag{6.34}$$

and if $Q > 0$ then

$$P > 0. \tag{6.35}$$

These results state that the error covariance matrix, $P_{k|k-1}$, converges to a finite, positive definite constant matrix when the system in (6.1) is controllable and observable. Consequently, the error in the estimate $\hat{x}_{k|k-1}$ is bounded as $k \to \infty$ since $P < \infty$. Furthermore, because $P > 0$ the Kalman filter gain is also positive definite. Consequently, the new observations are always included in the new state estimate.

Another consequence of steady-state analysis of the Kalman filter is that one can use the steady-state Kalman gain in place of the time varying Kalman gain. The advantage of such an approach is that considerable computational savings are possible due to the fact that one is not re-calculating the Kalman gain for each new observation. It is important to realize, however, that using the steady-state Kalman gain does not yield the optimal estimate of the state until after the transient behavior of the filter has died away.

To use the steady-state Kalman gain to implement the Kalman filter algorithm there exists, in Scilab, a function and a primitive which when used in tandem yield the estimated state. The function is `optgain` and the primitive is `ltitr`. The function `optgain` calculates the steady-state error covariance matrix and the steady-state Kalman gain given the system matrices and the noise covariance matrices. The primitive `ltitr` generates the state vectors from a linear dynamic system in state space form given the system matrices of this system and the input vectors. In the case of the Kalman filter, the input vectors are the observations y_k and the system dynamics are

$$\hat{x}_{k|k} = (I - KH)F\hat{x}_{k-1|k-1} + Ky_k \tag{6.36}$$

The following section describes how to use `sskf` the steady-state Kalman filter function. This is followed by an example using `sskf`. The following section describes the use of `kalm` the time-varying version of the Kalman filter function. Finally, several sections follow describing the square-root Kalman filter algorithm and how to use the associated function `srkf`.

6.1.6 How to Use the Macro `sskf`

The syntax of the `sskf` is as follows

```
[xe,pe]=sskf(y,f,h,q,r,x0)
```

where the system is assumed to take the following form

$$\begin{aligned} x_{k+1} &= \mathtt{f}x_k + w_k \\ y_k &= \mathtt{h}x_k + v_k \end{aligned}$$

where

$$\begin{aligned} w_k &\sim N(0,\ \mathtt{q}) \\ v_k &\sim N(0,\ \mathtt{r}) \\ x_0 &\sim N(\ \mathtt{x0}, \Pi_0). \end{aligned}$$

The remaining input, $\mathtt{y} = [y_0, y_1, \ldots, y_n]$, is a matrix where the individual observations y_k are contained in the matrix as column vectors.

The outputs `xe` and `pe` are the estimated state and the steady-state error covariance matrix. The form of the estimates is $\mathtt{xe} = [\hat{x}_{0|0}, \hat{x}_{1|1}, \ldots, \hat{x}_{n|n}]$ where each $\hat{x}_{k|k}$ is a column in the matrix. The steady-state error covariance matrix is a square matrix of the dimension of the state.

6.1.7 An Example Using the `sskf` Macro

The example uses the following system model and prior statistics:

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1.1 & .1 \\ 0 & .8 \end{bmatrix} x_k + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} w_k \\ y_k &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + v_k \end{aligned}$$

where

$$\begin{aligned} E\{w_k w_k^T\} &= \begin{bmatrix} .03 & .01 \\ .01 & .03 \end{bmatrix} \\ E\{v_k v_k^T\} &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

and

$$\begin{aligned} E\{x_0\} &= \begin{bmatrix} 10 \\ 10 \end{bmatrix} \\ E\{(x_0 - m_0)(x_0 - m_0)^T\} &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

Observations from the above system were generated synthetically using the primitive `ltitr`. The results of the steady-state Kalman filter estimates are shown in Figure ?? `exec('kf1.code')` Steady-State Kalman Filter Trackingf.kf0 where the dotted line marked by stars indicates the actual state path and the solid line marked by circles marks the estimated state path.

6.1.8 How to Use the Function `kalm`

The function `kalm` takes as input the system description matrices, the statistics of the noise processes, the prior estimate of the state and error covariance matrix, and the new observation. The outputs are the new estimates of the state and error covariance matrix. The call to `kalm` is as follows:

```
--> [x1,p1,x,p]=kf(y,x0,p0,f,g,h,q,r)
```

where `y` is the new observation, `x0` and `p0` are the prior state and error covariance estimates at time $t = 0$ based on observations up to time $t = -1$, `f`, `g` and `h` are the dynamics, input, and observation matrices, respectively, and `q` and `r` are the noise covariance matrices for the dynamics and observations equations, respectively. The outputs `x1` and `p1` are the new state and error covariance estimates at time $t = 1$ given observations up to time $t = 0$, respectively, and `x` and `p` are the new state and error covariance estimates at time $t = 0$ given observations up to time $t = 0$, respectively.

6.1.9 Examples Using the `kalm` Function

Three examples are illustrated in this section. All three examples are for two-dimensional state vectors. The first example illustrates Kalman tracking for a system model which is controllable and observable. The second and third examples show the results of uncontrollable and unobservable system models, respectively.

The first example uses the following system model and prior statistics:

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1.1 & .1 \\ 0 & .8 \end{bmatrix} x_k + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} w_k \\ y_k &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + v_k \end{aligned}$$

where

$$\begin{aligned} E\{w_k w_k^T\} &= \begin{bmatrix} .03 & .01 \\ .01 & .03 \end{bmatrix} \\ E\{v_k v_k^T\} &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

and

$$\begin{aligned} E\{x_0\} &= \begin{bmatrix} 10 \\ 10 \end{bmatrix} \\ E\{(x_0 - m_0)(x_0 - m_0)^T\} &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

Observations from the above system were generated synthetically using the system formulation and values from a random number generator for the dynamics and observations noise. These observations were then used as input to the Kalman filter. The result of ten observations is illustrated in Figure **??**. `exec('kf2.code')` Kalman Filter Trackingf.kf1 The figure shows the actual state path and the Kalman estimate of the state path as a dotted and solid line, respectively. The actual locations of the state and estimated values are indicated by the star and circle symbols on the respective paths. The ellipses in the figure are centered about the positions of the actual state path and their borders represent two standard deviations of estimation error calculated from the error covariance matrices. The values of the standard deviations for the above example are displayed below:

Each row of the above vector represents the standard deviations of the state vector error covariance matrix where the first row is the standard deviation of the *a priori* error in the state vector and the last row is the standard deviation of the state vector estimate at the last step of the Kalman filter. The above standard deviation vector is instructive. It should be noted that for both state values, the standard deviation is converging to a steady state value. For the first element of the state vector this value is .7800312 and for the second element of the state vector the value is .2824549. The convergence is to be expected since the above formulation of the system is both controllable and observable.

If we were to change the above system formulation so that the dynamics equation were now

$$x_{k+1} = \begin{bmatrix} 1.1 & .1 \\ 0 & .8 \end{bmatrix} x_k + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} w_k \tag{6.37}$$

the system would be uncontrollable. Re-running the Kalman filter with this re-formulation yields the following sequence of standard deviations:

As can be seen, the second state variable has a standard deviation which is not converging to a positive value. In fact the value of this standard deviation converges to zero. As was discussed in the section on the asymptotic behavior of the Kalman filter, this is what was to be expected. The result of this behavior is that the Kalman filter ignores any observed information regarding the second state variable since the error variance is going to zero (and, thus, the filter thinks that it has perfect information concerning this state value). If the above model is perfectly accurate then such an eventuality is not a problem. However, in practice there are modeling errors and, consequently, if the new observations are ignored, there is a danger that the Kalman filter estimates will begin to diverge from the actual state of the process.

Now we change the original model formulation again so that the observation equation is now

$$y_k = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} x_k + v_k. \tag{6.38}$$

Under these conditions the system is not observable. The evolution of the standard deviation for this example is:

Here the standard deviation of the first state variable is growing without bound. This is due to two things. First, the system is unobservable in the first state variable. Consequently, the observations provide no useful information concerning the estimate of this state. Secondly, since the system matrix `f` has un unstable eigenvalue the standard deviation of this state, in the limit, is unbounded.

The Scilab code used to generate the examples in this section is displayed below. The code for the steady-state Kalman filter example is as follows:

The code used to generate the non-steady-state Kalman filter example is:

6.2 The Square Root Kalman Filter

The Kalman filter is known to have certain numerical instabilities [3]. For example, since the update of the error covariance matrix involves taking differences of matrices (see(6.28)) it is possible that machine discretization error could result in a non positive semi-definite error covariance. Such an event could lead to severe divergence of the Kalman filter because this could render the Kalman gain infinite. Fortunately, there are a class of algorithms which are known collectively as square root Kalman filters which avoid this problem.

The square root Kalman filter propagates the state estimate and the square root of the error covariance matrix. (Here, S is defined to be the square root of a matrix, A, if $A = SS^T$. The square root, S, is not unique. For example for any other orthogonal matrix, O, such that $OO^T = I$ it follows that SO is also a square root of A. We denote the square root of A by $A^{1/2}$). The advantage of propagating the square root of the error covariance matrix is two-fold. First, the error covariance is always positive semi-definite since it is formed by squaring the square root. Second, the dynamic range of the elements in the square root of the error covariance is much smaller than that in the error covariance matrix itself. Consequently, the accuracy of calculations is improved since the machine precision is better adapted to representing the dynamic range of the square root. The details of a square root Kalman filter algorithm base on using Householder transformations is detailed below.

Restating the model for the dynamics and observations of the process in (6.1), we have that

$$\begin{aligned} y_k &= H_k x_k + v_k \\ x_{k+1} &= F_k x_k + w_k \end{aligned} \tag{6.39}$$

where w_k and v_k are independent zero-mean Gaussian vectors of covariance Q_k and R_k, respectively. The model in (6.39) is modified so that $w_k = G_k\mu_k$ and $v_k = L_k\mu_k$ where μ_k is a zero-mean Gaussian random vector with unit covariance matrix (i.e., $E\{\mu_k\mu_k^T\} = I$). Since w_k and $v_k$are independent $G_kL_k^T = L_kG_k^T = 0$. Now, the model in (6.39) can be expressed as

$$\begin{bmatrix} y_k \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} H_k & L_k \\ F_k & G_k \end{bmatrix} \begin{bmatrix} x_k \\ \mu_k \end{bmatrix}. \tag{6.40}$$

Also, we recall from (6.7) and (6.8) that

$$\begin{bmatrix} \hat{y}_{k|k-1} \\ \hat{x}_{k+1|k-1} \end{bmatrix} = \begin{bmatrix} H_k \\ F_k \end{bmatrix} \hat{x}_{k|k-1}. \tag{6.41}$$

Substracting (6.41) from (6.40) yields

$$\begin{bmatrix} \nu_k \\ \epsilon_{k+1|k-1} \end{bmatrix} = \begin{bmatrix} H_k P^{1/2}_{k|k-1} & L_k \\ F_k P^{1/2}_{k|k-1} & G_k \end{bmatrix} \begin{bmatrix} P^{-1/2}_{k|k-1}\epsilon_{k|k-1} \\ \mu_k \end{bmatrix} \tag{6.42}$$

where $\nu_k = y_k - H\hat{x}_{k|k-1}$, $\epsilon_{k+1|k-1} = x_{k+1} - \hat{x}_{k+1|k-1}$, $\epsilon_{k|k-1} = x_k - \hat{x}_{k|k-1}$, and where we have used the square root of the error covariance, $P_{k|k-1}$, so that the vector $P^{-1/2}_{k|k-1}\hat{x}_{k|k-1}$ would have unit covariance.

Now we assume that a matrix, T_k, exists such that $T_k T_k^T = I$ and such that

$$\begin{bmatrix} H_k P^{1/2}_{k|k-1} & L_k \\ F_k P^{1/2}_{k|k-1} & G_k \end{bmatrix} T_k = \begin{bmatrix} A_k & 0 \\ B_k & C_k \end{bmatrix}. \tag{6.43}$$

(The matrix T_k can always be constructed as a combination of Householder transformations. The Householder transformation and the construction of T_k are detailed in the next section).

Using T_k, (6.42) can be rewritten so that

$$\begin{aligned} \begin{bmatrix} \nu_k \\ \epsilon_{k+1|k-1} \end{bmatrix} &= \begin{bmatrix} H_k P^{1/2}_{k|k-1} & L_k \\ F_k P^{1/2}_{k|k-1} & G_k \end{bmatrix} T_k T_k^T \begin{bmatrix} P^{-1/2}_{k|k-1}\epsilon_{k|k-1} \\ \mu_k \end{bmatrix} \\ &= \begin{bmatrix} A_k & 0 \\ B_k & C_k \end{bmatrix} \eta_k \end{aligned} \tag{6.44}$$

where η_k is a zero-mean Gaussian random vector with unit covariance. We can now derive the significance of (6.44) by calculating the conditional mean and conditional covariance of $\epsilon_{k+1|k-1}$ given ν_k. Using (6.5) we obtain

$$\begin{aligned} E\{\epsilon_{k+1|k-1}|\nu_k\} &= E\{x_{k+1} - \hat{x}_{k+1|k-1}|\nu_k\} \\ &= \Lambda_{\epsilon_{k+1|k-1}\nu_k}\Lambda^{-1}_{\nu_k}\nu_k \end{aligned} \tag{6.45}$$

and

$$\begin{aligned} &E\{[\epsilon_{k+1|k-1} - E\{\epsilon_{k+1|k-1}\}][\epsilon_{k+1|k-1} - E\{\epsilon_{k+1|k-1}\}]^T|\nu_k\} = \\ &\Lambda_{\epsilon_{k+1|k-1}} - \Lambda_{\epsilon_{k+1|k-1}\nu_k}\Lambda^{-1}_{\nu_k}\Lambda_{\epsilon_{k+1|k-1}\nu_k} \end{aligned} \tag{6.46}$$

Calculating the covariances in (6.45) and (6.46) we obtain (using (6.44))

$$\begin{aligned} \Lambda_{\epsilon_{k+1|k-1}\nu_k} &= B_k A_k^T \\ \Lambda_{\nu_k} &= A_k A_k^T \\ \Lambda_{\epsilon_{k+1|k-1}} &= B_k B_k^T + C_k C_k^T. \end{aligned} \tag{6.47}$$

It can be seen that (6.45) yields the Kalman filter equations. Consequently, using (6.47) we obtain the Kalman gain, K_k, and the updated square root of the error covariance, $P^{1/2}_{k+1|k}$, as

$$\begin{aligned} K_k &= B_k A_k^{-1} \\ P^{1/2}_{k+1|k} &= C_k. \end{aligned} \tag{6.48}$$

The square root Kalman filter algorithm works as follows. Given $\hat{x}_{k|k-1}$, $P^{1/2}_{k|k-1}$, and y_k we first form the matrix on the left hand side of (6.43). The matrices G_k and L_k are obtained by performing a Cholesky decomposition of Q_k and R_k (that is $Q_k = G_k G_k^T$ and $R_k = L_k L_k^T$ where G_k and L_k are upper triangular). Using the Householder transformation T_k we obtain A_k, B_k, and C_k. Then, the updates are calculated by

$$\begin{aligned} \hat{x}_{k+1|k} &= F_k \hat{x}_{k|k-1} + B_k A_k^{-1}(y_k - H_k \hat{x}_{k|k-1}) \\ P_{k+1|k}^{1/2} &= C_k. \end{aligned} \tag{6.49}$$

All that remains is specification of the Householder transformation which is done in the following section.

6.2.1 The Householder Transformation

Let β be a vector in R^N. Then, we define the $N \times N$ matrix, T_β, such that

$$T_\beta = I - \frac{2}{\beta^T \beta} \beta \beta^T. \tag{6.50}$$

The matrices defined by (6.48) are called Householder transformations (see [29]). The T_β have the following properties

$$\begin{aligned} T_\beta &= T_\beta^T \\ T_\beta T_\beta^T &= I \\ T_\beta T_\beta &= I. \end{aligned} \tag{6.51}$$

Furthermore, the matrix T_β has a geometric interpretation. The vector β uniquely defines a hyperplane in R^N as the sub-space in R^N which is orthogonal to β. The matrix T_β acts as a reflector in that T_β maps vectors in R^N to their reflected images through the hyper-plane defined by β.

Householder transformations can be used to upper triangularize matrices. This is accomplished as follows. Let $A = [a_1, a_2, \cdots, a_M]$ be an $N \times M$ matrix where a_k is an N-vector and let T_{β_1} be a Householder transformation where β_1 is constructed such that

$$T_{\beta_1} a_1 = \tilde{a}_1 = \begin{bmatrix} \sqrt{\sum_{k=1}^N a_{1k}^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.52}$$

(i.e., we find the β_1 such that when a_1 is reflected through the hyper-plane defined by β_1 its image is as above). The choice for β_1 which accomplishes this goal is

$$\beta_1 = \begin{bmatrix} a_{11} + \sqrt{\sum_{k=1}^N a_{1k}^2} \\ a_{12} \\ \vdots \\ a_{1N} \end{bmatrix} \tag{6.53}$$

where a_{1k} is the k^{th} element of the N-vector a_1.

The Householder transformation, T_{β_1}, can be used now to transform the matrix A,

$$T_{\beta_1} A = [\tilde{a}_1, \tilde{a}_2, \cdots, \tilde{a}_M] \tag{6.54}$$

where $\tilde{a}_1$ is as in (6.52) and the remaining columns of A are transformed to $\tilde{a}_k = T_{\beta_1} a_k$.

Now we construct a second Householder transformation, T_{β_2} such that

$$T_{\beta_2}\tilde{a}_2 = \tilde{\tilde{a}}_2 = \begin{bmatrix} \tilde{a}_{21} \\ \sqrt{\sum_{k=2}^{N}\tilde{a}_{2k}^2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.55}$$

That is, the first element of $\tilde{a}_2$ is left unchanged, the second element becomes the norm of the rest of the elements and the remaining elements become zero. The choice for β_2 which accomplishes this is

$$\beta_2 = \begin{bmatrix} 0 \\ \tilde{a}_{22} + \sqrt{\sum_{k=2}^{N}\tilde{a}_{2k}^2} \\ \tilde{a}_{23} \\ \vdots \\ \tilde{a}_{2N} \end{bmatrix}. \tag{6.56}$$

Calculating $T_{\beta_2}T_{\beta_1}A$ yields

$$T_{\beta_2}T_{\beta_1}A = [\tilde{a}_1, \tilde{\tilde{a}}_2, \tilde{\tilde{a}}_3, \cdots, \tilde{\tilde{a}}_M] \tag{6.57}$$

where $\tilde{a}_1$ is still as in (6.52) and $\tilde{\tilde{a}}_2$ is as in (6.55). Continuing recursively in this way we can upper triangularize the A matrix.

The Scilab primitive `qr` performs a triangularization of a matrix using Householder transformations. In particular, it is the r part of the qr-decomposition which yields the desired triangularized matrix.

6.2.2 How to Use the Macro `srkf`

The function `srkf` takes as input the system description matrices, the statistics of the noise processes, the prior estimate of the state and error covariance matrix, and the new observation. The outputs are the new estimates of the state and error covariance matrix. The call to `kalm` is as follows:

```
--> [x1,p1]=srkf(y,x0,p0,f,h,q,r)
```

where `y` is the new observation, `x0` and `p0` are the prior state and error covariance estimates, `f` and `h` are the dynamics and observation matrices, respectively, and `q` and `r` are the noise covariance matrices for the dynamics and observations equations, respectively. The outputs `x1` and `p1` are the new state and error covariance estimates, respectively.

6.3 The Wiener Filter

The generalized Wiener filtering problem [14] can be posed as follows. It is desired to estimate a zero-mean, Gaussian vector process, $x(t)$, in the interval $[a, b]$ from observations of a statistically related, zero-mean, Gaussian vector, $y(t)$. Furthermore, the covariance functions $R_{yy}(t, s) = E\{y(t)y^T(s)\}$ and $R_{xy}(t, s) = E\{x(t)y^T(s)\}$ are assumed known for $t, s \in [a, b]$. Then, the least mean squared error estimate of $x(t)$ given $y(t)$ is obtained as a linear operator on $y(t)$ as

$$\hat{x}(t) = \int_a^b H(t, s)y(s)ds \tag{6.58}$$

where $\hat{x}(t)$ denotes the least mean squared estimate and $H(t,s)$ is an $N \times M$ matrix function of two arguments t and s (i.e., $[H(t,s)]_{ij} = h_{ij}(t,s)$ and $\hat{x}_i(t) = \int_a^b \sum_{j=1}^M h_{ij}(t,s)y_j(s)ds$).

By the principle of orthogonality the error in the estimate in (6.58), $x(t) - \hat{x}(t)$, must be orthogonal to $y(u)$ for $t, u \in [a,b]$. Thus,

$$0 = E\{(x(t) - \hat{x}(t))y^T(u)\} = R_{xy}(t,u) - \int_a^b H(t,s)R_{yy}(s,u)ds. \tag{6.59}$$

Solving the matrix integral equation in (6.59) for $H(t,s)$ yields the optimal estimator for $x(t)$ when used in (6.58). A general solution of (6.59) is not possible, however, for some very important special cases (6.59) is resolvable. For example, if the covariance functions R_{xy} and R_{yy} are wide sense stationary and with rational spectral densities then specific solutions techniques are available.

The sections which follow address Wiener filtering for a special formulation of the relationship between $x(t)$ and $y(t)$ (albeit, a relationship for which many engineering problems of interest can be formulated) and, consequently, permits a solution to the problem posed by (6.58) and (6.59). This special formulation consists of a state space difference equation identical to that used in the development of the Kalman filter. This realization has some advantages in that it allows finite interval estimation of non-stationary processes. The disadvantage of this procedure is that sometimes the only knowledge of the processes x and y is their covariance structures. In this case, the construction of a state-space model which has the appropriate covariance characteristics may not always be readily evident.

6.3.1 Problem Formulation

In our problem formulation it is assumed that a dynamic model of the process x is known, that the process y is related to the process x through an observation equation, and that both processes are discrete. Consequently, we have the equations

$$\begin{aligned} x_{k+1} &= F_k x_k + G_k u_k \\ y_k &= H_k x_k + v_k \end{aligned} \tag{6.60}$$

where x_0, u_k, and v_k are independent and Gaussian random vectors with statistics

$$\begin{aligned} u_k &\sim N(0, Q_k) \\ v_k &\sim N(0, R_k) \\ x_0 &\sim N(m_0, \Pi_0) \end{aligned} \tag{6.61}$$

From the formulation in (6.60) and (6.61) one can determine the covariance functions R_{xy} and R_{yy} and, thus, it is reasonable to try and solve a discrete version of (6.59) to obtain the optimal filter. However, there are certain aspects of the dynamics and observations models which permit a solution to the problem posed in the previous section without explicitly solving for the filter designated by (6.59).

The solution to the discrete version of (6.58) based on the modelization of the underlying processes given by (6.60) and (6.61) can be achieved by several different Kalman filter formulations. The particular approach used here is known as the Rauch-Tung-Striebel formulation [10]. In this approach a Kalman filter is used on the data in the usual way to obtain estimates $\hat{x}_{k|k-1}$ and $\hat{x}_{k|k}$ along with their associated error covariances $P_{k|k-1}$ and $P_{k|k}$. A second recursive filter is then used on the estimates and their error covariances to obtain the estimates $\hat{x}_{k|N}$ and their error covariances $P_{k|N}$ (that is, the estimate and error covariance of x_k based on all the data in the interval $[0, N]$). This second filter processes the estimates in the backward direction using $\hat{x}_{N|N}$ and $P_{N|N}$ to initialize the filter, then using $\hat{x}_{N-1|N-1}$, $\hat{x}_{N-1|N-2}$, $P_{N-1|N-1}$, and $P_{N-1|N-2}$ to obtain $\hat{x}_{N-1|N}$ and $P_{N-1|N}$, etc., continuing recursively in this way over the entire interval.

A complete development of the Rauch-Tung-Striebel approach is too involved to recount here. We outline a procedure which may be used to obtain the result without discussing all the details. The interested reader can find a complete and unified development in [20].

The approach we take is based on a constrained maximum likelihood solution to the problem. We form a cost function, $J(u, v)$, which we desire to minimize with respect to its arguments. The function J is defined by

$$J(u, v) = u^T Q^{-1} u + v^T R^{-1} v \tag{6.62}$$

where

$$\begin{aligned}
u &= \begin{bmatrix} m_0 \\ u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \\
v &= \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_N \end{bmatrix} \\
Q &= \begin{bmatrix} \Pi_0 & 0 & \cdots & 0 \\ 0 & Q_0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & Q_{N-1} \end{bmatrix} \\
R &= \begin{bmatrix} R_0 & 0 & \cdots & 0 \\ 0 & R_1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & R_N \end{bmatrix}
\end{aligned} \tag{6.63}$$

and where m_0 and Π_0 are the *a priori* statistics of x_0. The functional in (6.62) is minimized subject to the constraints

$$\begin{aligned}
Sx &= Gu \\
y &= Hx + v
\end{aligned} \tag{6.64}$$

where

$$\begin{aligned}
x &= \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} \\
y &= \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
S &= \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -F_0 & I & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & -F_{N-1} & I \end{bmatrix} \\
G &= \begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & G_0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & G_{N-1} \end{bmatrix} \\
H &= \begin{bmatrix} H_0 & 0 & \cdots & 0 \\ 0 & G_1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & H_N \end{bmatrix}.
\end{aligned} \tag{6.65}$$

Minimizing (6.62) subject to the system constraints in (6.64) yields the estimate $\hat{x}^T = [\hat{x}_{0|N}^T, \hat{x}_{1|N}^T, \ldots, \hat{x}_{N|N}^T]^T$ The optimization can be accomplished using the Lagrange multiplier technique where a new functional $\tilde{J}$ is formed from (6.62) and (6.64)

$$\tilde{J}(u,v) = J(u,v) + \lambda^T(Sx - Gu) \tag{6.66}$$

where λ, an unknown vector of the same dimension as x, is called the Lagrange multiplier. After some algebra and setting the partial derivatives of $\tilde{J}$ with respect to u and x to zero we obtain the so-called Hamiltonian (see [15])

$$\begin{bmatrix} S & -GQG^T \\ H^TR^{-1}H & S^T \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ H^TR^{-1}y \end{bmatrix}. \tag{6.67}$$

Solution of (6.67) yields $\hat{x}_{k|N}$ for $k = 0, 1, \ldots, N$. It can be seen that (6.67) is a sparse block matrix. In fact, after a re-ordering of the terms in the vectors, the resulting matrix is tri-block-diagonal. Solving this tri-block-diagonal system by using Gaussian elimination, first on the lower diagonal, then on the upper diagonal yields the following equations in matrix form

$$\begin{bmatrix}
-\Pi_0 & I & 0 & 0 & 0 & \cdots & 0 \\
I & \Theta_0 & -F_0^T & 0 & 0 & \cdots & 0 \\
0 & -F_0 & \Gamma_0 & I & 0 & \cdots & 0 \\
0 & 0 & I & \Theta_1 & -F_1^T & \cdots & 0 \\
 & & & \cdots\cdots\cdots & & & \\
 & & & \cdots\cdots\cdots & & & \\
 & & & \cdots\cdots\cdots & & & \\
0 & 0 & \cdots & 0 & -F_{N-1} & \Gamma_{N-1} & I \\
0 & 0 & \cdots & 0 & 0 & I & \Theta_N
\end{bmatrix}
\begin{bmatrix} \hat{\lambda}_0 \\ \hat{x}_0 \\ \hat{\lambda}_1 \\ \hat{x}_1 \\ \vdots \\ \hat{\lambda}_N \\ \hat{x}_N \end{bmatrix}
=
\begin{bmatrix} m_0 \\ \psi_0 \\ 0 \\ \psi_1 \\ 0 \\ \vdots \\ \psi_N \end{bmatrix} \tag{6.68}$$

where $\Theta_k = H_k^TR_k^{-1}H_k$, $\Gamma_k = -G_kQ_kG_k^T$, and $\psi_k = H_kR_k^{-1}y_k$. Beginning by eliminating the lower diagonal of the matrix we obtain for the first equation

$$\hat{x}_0 = \Pi_0\hat{\lambda}_0 + m_0 \tag{6.69}$$

which when $\hat{\lambda}_0 = 0$ yields the usual Kalman filter initial condition $\hat{x}_{0|-1} = m_0$. The second equation obtained by the Gaussian elimination is

$$\hat{x}_0 = (H_0^TR_0^{-1}H_0 + \Pi_0^{-1})^{-1}(F_0^T\hat{\lambda}_1 + H_0R_0{-}1y_0 + \Pi_0^{-1}m_0) \tag{6.70}$$

which, when $\hat{\lambda}_1 = 0$ and after some algebra, yields the Kalman filter filter-update equation $\hat{x}_{0|0} = \hat{x}_{0|-1} + P_{0|-1}H_0^T(H_0P_{0|-1}H_0^T + R_0)^{-1}(y_0 - H_0\hat{x}_{0|-1})$. Continuing in this way the Gaussian elimination yields the $2N$ equations

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1}(y_k - H_k\hat{x}_{k|k-1}) \\ \hat{x}_{k+1|k} &= F_k\hat{x}_{k|k} \end{aligned} \tag{6.71}$$

where

$$\begin{aligned} P_{k|k} &= P_{k|k-1} - P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1}H_kP_{k|k-1} \\ P_{k+1|k} &= F_kP_{k|k}F_k^T + G_kQ_kG_k^T. \end{aligned} \tag{6.72}$$

After eliminating the lower diagonal of the matrix in (6.68) the upper diagonal is eliminated in the same way, however, now the $\hat{\lambda}_k$ are not set to zero which results in the equations

$$\hat{x}_{k|N} = \hat{x}_{k|k} + A_k[\hat{x}_{k+1|N} - \hat{x}_{k+1|k}] \tag{6.73}$$

where

$$\begin{aligned} P_{k|N} &= P_{k|k} + A_k[P_{k+1|N} - P_{k|k-1}]A_k^T \\ A_k &= P_{k|k}F_k^TP_{k|k-1}^{-1} \end{aligned} \tag{6.74}$$

and where the $\hat{\lambda}_k$ have been identified as the $\hat{x}_{k|N}$. It can be seen that equations (6.71) and (6.72) specify the standard Kalman filter. Equations (6.73) and (6.74) yield the Rauch-Tung-Striebel smoother.

6.3.2 How to Use the Function `wiener`

The syntax for the function `wiener` is as follows

```
-->[xs,ps,xf,pf]=wf(y,x0,p0,f,g,h,q,r)
```

The inputs `f`, `g`, and `h` are the system matrices in the interval $[t_0, t_f]$. The construction of, for example, `f` is accomplished by arranging the individual matrices F_k into a block-row matrix $[F_0, F_1, \ldots, F_N]$. The other system matrices are identical in construction. The inputs `q` and `r` are the covariance matrices of dynamics and observation noise. These matrices are also constructed in a fashion identical to that for `f`. The inputs `x0` and `p0` are the initial state estimate vector and error variance matrix. The vector `x0` must be in column form. Finally, the input `y` are the observations in the interval $[t_0, t_f]$. The form of `y` is a matrix where the first column is the observation y_0, the second column y_1, etc. The outputs are the smoothed estimates `xs` of x_0, x_1, etc. arranged in a matrix of similar form to `y` and `ps` which is arranged identically to the form of the input matrices.

6.3.3 Example

In this section, the `wiener` function is demonstrated on a two-dimensional problem. The input to the filter is synthetic data obtained using the same system dynamics given to the `wiener` function. The system used in the example is as follows.

$$\begin{aligned} x_{k+1} &= \begin{bmatrix} 1.15 & .1 \\ 0 & .8 \end{bmatrix} x_k + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} w_k \\ y_k &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + v_k \end{aligned}$$

where

$$\begin{aligned} E\{w_k w_k^T\} &= \begin{bmatrix} .01 & 0 \\ 0 & .01 \end{bmatrix} \\ E\{v_k v_k^T\} &= \begin{bmatrix} 20 & 0 \\ 0 & 20 \end{bmatrix} \end{aligned}$$

and

$$\begin{aligned} E\{x_0\} &= \begin{bmatrix} 10 \\ 10 \end{bmatrix} \\ E\{(x_0 - m_0)(x_0 - m_0)^T\} &= \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \end{aligned}$$

Figure **??** shows the results of using the `wiener` function on the synthetic data generated by the model above. Here the dotted line indicates the actual state values generated by the system. The circles on the dotted line serve to mark the actual state locations. The solid line marked by circles indicates the Kalman filter estimate of the state. The estimate of x_0 is located in the upper left corner of the figure and the estimate of x_{12} is located in the lower right corner of the figure. As can be seen the initial estimates obtained by the Kalman filter are not so good with respect to the final estimates obtained by the filter. This is due to the large initial error covariance matrix given for the initial estimate of the state vector. The solid line marked by stars is the smoothed Kalman filter estimate. As can be seen, the final smoothed estimate is identical to that of the regular Kalman filter. However, as the smoothed Kalman filter estimate works its way back to the initial state value, the estimate becomes greatly improved, as is to be expected since these states are now estimated based on all of the observed data. `exec('wf1.code')` Wiener Smoothing Filterf.wf1

The Scilab code which generated the example in this section is as follows.

Chapter 7

Optimization in filter design

In this chapter, some optimization techniques are investigated for the design of IIR as well as FIR filters. Those techniques are particuliarly usefull when non standard responses are desired.

7.1 Optimized IIR filters

In a previous chapter on the design of IIR filters, several methods have been investigated which make use of closed-form expressions for this purpose. In this section the desired specifications are achieved with the help of optimization techniques: the set of convenient filter parameters is computed by minimizing some error criterion [24]. This technique has to be used when nonconventional filter specifications are to be achieved.

7.1.1 Minimum Lp design

The criterion that has been retained here is the minimum $\mathbb{L}_p$ error and the set of parameters to be optimized, the set of poles and zeros the cardinal of which being specified by the user. The reason for such a choice of parameters is that specifications on the group delay are much more easily written for this set than for the usual filter coefficients - especially the computations of gradients - One may note also that the minimum $\mathbb{L}_p$ error criterion admits the well-known minimum mean-square error criterion as a particular case by setting p to two.

Now, let $H(z)$ be the transfer function of the filter given as a cascade of K second-order sections:

$$H(z) = A \prod_{k=1}^{K} \frac{z^2 - 2r_{0k}cos\theta_{0k}z + r_{0k}^2}{z^2 - 2r_{pk}cos\theta_{pk}z + r_{pk}^2} \tag{7.1}$$

The set of parameters to be optimized is then given by the following vector:

$$\phi = (r_{0k}, \theta_{0k}, r_{pk}, \theta_{pk}, A) \quad k = 1, K \tag{7.2}$$

where index 0 stands for zeros and index p for poles, no confusion being to be expected with index p in the $\mathbb{L}_p$ error criterion. Usually the specifications for a filter are given separately for the magnitude $|H(e^{j\omega})|$ and/or the group delay $\tau(\omega)$; the corresponding expressions are:

$$|H(e^{j\omega})| \triangleq a(\phi, \omega) \tag{7.3}$$

$$= A \prod_{k=1}^{K} \frac{(1 - 2r_{0k}cos(\omega - \theta_{0k}) + r_{0k}^2)^{1/2}(1 - 2r_{0k}cos(\omega + \theta_{0k}) + r_{0k}^2)^{1/2}}{(1 - 2r_{pk}cos(\omega - \theta_{pk}) + r_{pk}^2)^{1/2}(1 - 2r_{pk}cos(\omega + \theta_{pk}) + r_{pk}^2)^{1/2}} \tag{7.4}$$

and

$$\begin{aligned}&\tau(\phi,\omega) = \\ &\sum_{k=1}^{K}\{\frac{1-r_{pk}cos(\omega-\theta_{pk})}{(1-2r_{pk}cos(\omega-\theta_{pk})+r_{pk}^2)^{1/2}}+\frac{1-r_{pk}cos(\omega+\theta_{pk})}{(1-2r_{pk}cos(\omega+\theta_{pk})+r_{pk}^2)^{1/2}} \\ &-\frac{1-r_{0k}cos(\omega-\theta_{0k})}{(1-2r_{0k}cos(\omega-\theta_{0k})+r_{0k}^2)^{1/2}}-\frac{1-r_{0k}cos(\omega+\theta_{0k})}{(1-2r_{0k}cos(\omega+\theta_{0k})+r_{0k}^2)^{1/2}}\}\end{aligned} \tag{7.5}$$

Defining the desired magnitude response $a_d(\omega_j)$ and group delay $\tau_d(\omega)$, the minimum $\mathbb{L}_p$-design problem can be formulated by mean of the following error function:

$$\begin{aligned}E(\phi) &= \lambda\sum_{j=1}^{J}w_a(\omega_j)[a(\phi,\omega_j)-a_d(\omega_j)]^{2p} \\ &+ (1-\lambda)\sum_{j=1}^{J}w_\tau(\omega_j)[\tau(\phi,\omega_j)-\tau_d(\omega_j)]^{2p}\end{aligned} \tag{7.6}$$

where $w_a(\omega_j)$ and $w_\tau(\omega_j)$ are weighting functions defined on a dense grid of frequencies $\{\omega_j/0 \leq \omega_j \leq \pi\}$ and λ is a real number belonging to $[0,1]$ that reflects the importance of the specifications on the magnitude relative to that on the group delay in a straightforward fashion. One seek after a vector ϕ^* such that $E(\phi^*)$ is minimum: this problem is easily solved in Scilab with the help of the function `optim` the purpose of which is the resolution of general nonlinear optimization problems . We refer the reader to the relevant documentation [7] for a detailed explanation of its use. The `optim` function needs some input parameters, the main of which being what is called a *simulator*: it may be given as a Scilab function and its purpose is to give the cost function of the relevant problem as well as its gradient relative to the specified parameters. For the minimum $\mathbb{L}_p$ design problem, the simulator is named `iirlp` and makes use of two other macros: `iirmag` and `iirgroup`; it gives $E(\phi)$ together with its gradient relative to ϕ.

The following example will give an idea of what can be achieved with this optimization technique: we are given a low-pass elliptic filter with normalized cut-off frequency 0.25, transition bandwidth equal to 0.15, ripple in the passband of 0.1 and ripple in the stopband of 0.01 (i.e. 40dB of attenuation); with the Scilab function `eqiir` we have obtained a filter of the fourth order together with its zeros and poles.

Now we want to inverse this filter, that is we seek after a filter the magnitude reponse of which times that of the original elliptic filter equals one in the passband, while in the stopband the total attenuation is less than 80dB. This situation appears for example when a digital filter is needed, after analog-to-digital conversion, to compensate the deformations of a signal by an anti-aliasing analog filter. The corresponding specifications are obtained the following way:

Although the constraints on the parameters can be easily specified, an unconstrained optimization procedure has been applied because the initial values are unknown (hence may be far away from the optimal solution, if any) and the order is unknown too. Instead, as indicated in [23], the Scilab function `optim` will be applied several times and when some pole or zero goes outside the unit circle, it will be replaced by the symmetric (with respect to the unit circle) complex number and a new optimization runned. To see the results obtained with a constrained optimization, it may be interesting to run the following command, recalling that the constraints on the poles and zeros are:

$$\left\{ \begin{array}{ll} 0 \leq r_{0k} \leq 1 & 0 \leq r_{pk} \leq 1 \\ 0 \leq \theta_{0k} \leq 2\pi & 0 \leq \theta_{pk} \leq 2\pi \end{array} \right. \tag{7.7}$$

Another method to solve this problem would be to run an optimization with penalties on the constraints, in order to keep the poles and zeros inside the unit circle: we did not try it. Now, back to the unconstrained optimization, after several runs of `optim` without constraints, an optimal solution has been achieved for the chosen filter order. Nevertheless it is seen on Figure **??** that this solution is not satisfactory: `exec('optiir.1.code')` Minimum mean-square design. Fourth order IIR filterf.minlp.1 Figure **??** shows that the product of the two magnitude responses is far from being equal to one in the passband and that the total prescribed attenuation is not achieved (the horizontal line is at -80 dB): `exec('optiir.2.code')` Resulting magnitude response. Log scalef.minlp.2 So a second-order block has been added (four more parameters) to the transfer function found at the preceding step, leading to a filter of order six:

The same optimization procedure has been applied with this initial value, resulting in the following solution vector:

the desired magnitude reponse and the one achieved with that solution appear in Figure **??**, while the product of the log-magnitude responses is in Figure **??**. `exec('optiir.3.code')` Minimum mean-square design. Sixth order IIR filterf.minlp.3 `exec('optiir.4.code')` Resulting magnitude response. Log scalef.minlp.4 As we are interested in what happens in the passband, focusing on it is needed: this is done in Figure **??** and we see that for $\omega \in [0, .45]$ the ripple is equal to 0.07 dB. The reader may convince himself that better approximation may be obtained with an increase of the filter order; we mention too that the specification of a_d at the beginning of the transition is not likely to be that of a real filter (it has not a monotone decreasing behaviour in that region !) and that a more realistic desired response could have been best approximated with the same number of parameters. `exec('optiir.5.code')` Log-magnitude response. $\omega \in [0, 0.45]$f.minlp.5

7.2 Optimized FIR filters

As for IIR filters, optimization techniques may be used to achieve particuliar specifications for FIR filters [24] . In this framework, the design problem formulation leads to a simple linear programming problem, which makes this approach attractive, as opposed to nonlinear methods used in the case of optimization based, IIR filter design.

As the approach in this section is based on the frequency sampling technique 3.1, we first refer to the frequency response of a linear phase FIR filter as given by formula 3.1. In fact, because the linear phase term can be ignored for the design purpose, we rather consider the real function:

$$H^*(e^{j\omega}) \quad = \quad \sum_{k=0}^{N-1} H(k)S(\omega, k) \tag{7.8}$$

where

$$\begin{aligned} S(\omega,k) &= e^{-jk\pi/N}\frac{\sin(N\omega/2)}{\sin(\omega/2-k\pi/N)} \\ &= \pm e^{-jk\pi/N}\frac{\sin(N(\omega/2)-k\pi/N)}{\sin(\omega/2-k\pi/N)} \end{aligned} \tag{7.9}$$

are the interpolating functions. Usually in filter design, specifications are given in passbands and stopbands while absent in transition bands for which the width rather is given. For that reason, $H^*(e^{j\omega})$ can be written:

$$H^*(e^{j\omega}) = B(\omega) + \sum_{i=1}^{p} T_i A_i(\omega) \tag{7.10}$$

where $B(\omega)$ gives the contribution to $H^*(e^{j\omega})$ of all the fixed frequency samples (that is those in the passbands and the stopbands) and the $A_i(\omega)$ the contribution of all the unconstrained samples (that is the ones in the transitions) with respective magnitude T_i, these being to be optimized. In the sequel, the union of passbands will be called region 1,noted R_1 and that of passbands region 2, noted R_2. We now want, for a fixed approximation error in R_1, to find the linear phase FIR filter giving the maximum attenuation in R_2 - note that the converse is another possible issue - This can be formulated as follows:

For some fixed ϵ and desired frequency response $H_d(e^{j\omega})$, find the set of T_i, solution of:

$$\min_{T_i}\max_{\omega\in R_2} |H^*(e^{j\omega}) - H_d(e^{j\omega})| \tag{7.11}$$

and subject to the constraints:

$$|H^*(e^{j\omega}) - H_d(e^{j\omega})| \leq \epsilon \tag{7.12}$$

Because of the linearity of $H_d(e^{j\omega})$ with respect to the T_i, we are led to a linear programming problem, the cost function to minimize being the maximum attenuation in R_2, which will be denoted by T_{p+1} for notational convenience and the constraints on the $T_i, i = 1 \ldots p$, being formulated with the help of (7.12). The optimization problem may be formulated as to find the set of $T_i's$ such that T_{p+1} is minimum subject to the constraints:

$$\left.\begin{aligned} \textstyle\sum_{i=1}^{p} T_iA_i(\omega) &\leq \epsilon - B(\omega) + H_d(e^{j\omega}) \\ -\textstyle\sum_{i=1}^{p} T_iA_i(\omega) &\leq \epsilon + B(\omega) - H_d(e^{j\omega}) \end{aligned}\right\}\omega \in R_1$$

$$\left.\begin{aligned} \textstyle\sum_{i=1}^{p} T_iA_i(\omega) - T_{p+1} &\leq -B(\omega) + H_d(e^{j\omega}) \\ -\textstyle\sum_{i=1}^{p} T_iA_i(\omega) - T_{p+1} &\leq B(\omega) - H_d(e^{j\omega}) \end{aligned}\right\}\omega \in R_2$$

Now the problem is in a suitable form for solution via the classical *Simplex Method.* Let us mention too that, with minor changes, the problem -and the associated macro- may be stated as to find the filter with minimum ripple in the passbands for a given attenuation in the stopbands. In the following, only an example of the standard lowpass filter type is treated although any other frequency response can be approximated with this method.

Example 1 : figure **??** shows the frequency response of a lowpass type 1 filter with the following specifications: n=64; cut-off frequency, f_c=.15; $\epsilon = 0.01$; three samples in the transition. `exec('optfir1.code')` Linear programming design. 64-point lowpass FIR filteroptfir.fig1 Figure **??** shows the log magnitude squared of the initial filter defined by the rectangular window and the optimized filter. 0.28 and 0.30 $\epsilon = 0.001$ and three samples in each transition. `exec('optfir2.code')` Linear programming design. optfir.fig2

Chapter 8

Stochastic realization

Let y_k be a wide sense stationary gaussian process with covariance function $\{R_n; n \in \mathbb{Z}\}$. It is well-known that y_k may be considered as the output of a filter F with a white noise e_t as input. The *Stochastic Realization* problem for y_k is the construction of an algorithm giving F ; in addition, F is usually asked to be causal and minimum delay (i.e. with causal inverse), for reasons of realizability. One major consequence of this additional constraint is that F , when it exists, is minimal and unique, up to a transformation of coordinates. The filter F is called the *Modeling Filter* and its inverse F^{-1} the *Whitening Filter* . Let us consider the following examples, in which the information on y_k is given two different ways. First, let be given $S_y(z)$, the spectrum of y_k. Then, the whitening filter of y_k is the solution $E(z)$ of the *Spectral Factorization* problem of $S_y(z)$, that is the solution of :

$$E(z)S_y(z)E^T(z^{-1}) = I \tag{8.1}$$

such that $E(z)$ and $E^{-1}(z)$ are analytical in the unit disc, that is that the modeling filter of y_k is causal and minimum delay. The stochastic realization problem is then the computation of $E(z)$, given $S_y(e^{i\theta})$. Solutions to this problem will be given in section 8.1 with direct factorization of matrix polynomials and in 8.2 via a state-space approach.

Another example is when the covariance function $R_n = E(y_k y_{k-n}^T)$ of y_k is given - the information on y_k is then equivalent to that in the previous example- The whitening filter giving the innovation, or prediction error, is obtained by minimizing with respect to the coefficients A_k, the mean square prediction error :

$$E(\|e_t\|^2) = E\{\|y_k - \sum_{k>0} A_k y_{t-k}\|^2\} \tag{8.2}$$

The stochastic realization problem is then the computation of the A_k as the solution of the following *Yule-Walker* , normal equations :

$$\begin{bmatrix} R_0 & R_1 & R_2 \dots \\ R_1^T & R_0 & R_1 \dots \\ R_2^T & R_1^T & R_0 \dots \\ \vdots & & \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \end{bmatrix} \tag{8.3}$$

This system being Toëplitz, it may be efficiently solved using a Levinson-type algorithm ,as will be exposed in section 8.4.

8.1 The `sfact` primitive

Given a matrix polynomial $H(z)$ with size $n \times m$ and rank n, the Scilab function `sfact` gives the spectral factorization of $H(z)$, that is a matrix polynomial D such that:

$$H(z) = D(z)D(1/z)z^n \tag{8.4}$$

Consider for example the following 2×2 matrix polynomial $a(z)$, generated from the simulation of a two-dimensional process with three poles followed by a levinson filtering (see section 8.4):

8.2 Spectral Factorization via state-space models

As mentioned in the introduction, the stochastic realization problem may be formulated in terms of factorization of rational matrices - leading to factoring polynomials - or in a state-variable framework. In this section, we will concentrate on the state-space approach, with the underlying markovian representation following the treatment of [27] or that uses a gaussian spaces formulation of this problem. Thus, given y_k a zero-mean, stationary gaussian process, we seek after markovian models of the following type:

$$\begin{cases} x_{k+1} = Fx_k + Ju_{k+1} \\ y_{k+1} = Hx_k + Lu_{k+1} \end{cases} \tag{8.5}$$

with the following hypotheses:
x_k is q-dimensional.
u_k is a q-dimensional, standard, gaussian white noise.
F, H, J, L are arbitrary matrices with F asymptotically stable. Furthermore we shall restrict ourselves to minimal models, that is with F having minimum dimensions.

8.2.1 Spectral Study

Let $\{R_k; k \in \mathbb{Z}\}$ be the covariance function of y_k. Defining G as :

$$G = E(x_0 y_0^T) \tag{8.6}$$

we have that

$$\forall n \geq 1, \quad R_n = HF^{n-1}G \tag{8.7}$$

Letting $Y(z)$ and $U(z)$ be the z-transforms of y_k and u_k respectively, it follows from (8.5) that :

$$Y(z) = \Gamma(z)U(z) \tag{8.8}$$

where

$$\Gamma(z) = J + Hz(I - Fz)^{-1}L \tag{8.9}$$

is a rational matrix without poles in a vicinity of the unit disc. Thus, the spectrum S_y of y_k may be written in the following factored form:

$$S_y(\theta) = \Gamma(e^{i\theta})\Gamma^*(e^{-i\theta}) \quad \theta \in [-\pi, \pi] \tag{8.10}$$

where Γ^* denotes the transposed conjugate of Γ. Furthermore, from (8.8), we can write:

$$U(z) = \Gamma^{-1}(z)Y(z) \tag{8.11}$$

and when $J > 0$ (in the sense of positive matrices) :

$$\Gamma^{-1}(z) = J^{-1} - J^{-1}Hz(I - (F - LJ^{-1}H)z)^{-1}LJ^{-1} \tag{8.12}$$

so that u_k may be obtained from y_k by a Laurent expansion of Γ^{-1} in the vicinity of the unit circle, leading to the whitening filter .

It is worth noting that when y_k is scalar, then $\Gamma(z) = B(z)/A(z)$, where A and B are coprime polynomials and B has no zero in a vicinity of the unit disc; in other words, y_k is an ARMA process .

8.2.2 The Filter Model

Among the many markovian representations of y_k, one is of particular importance, as we shall see later on: The *Filter Model* or *Innovations Model* . To introduce it, we need some definitions: first, let us define the *filter* $\tilde{x}_k$ of x_k as:

$$\tilde{x}_k = E(x_k/Y_{k-1}^-) \tag{8.13}$$

where Y_k^- is the gaussian space generated by the coordinates of y_k, $k \leq n$. It is the filter of the process x_k by the process y_k. We need also the innovations process $\tilde{w}_k$ defined as follows :

$$\tilde{w}_k = y_k - E(y_k/Y_{k-1}^-) \tag{8.14}$$

$\tilde{w}_k$ is a stationary, gaussian white noise with covariance $\tilde{R}$. From $\tilde{w}_k$ the standard gaussian white noise $\tilde{u}_k$ may be obtained as: $\tilde{u}_k = \tilde{R}^{-1/2}\tilde{w}_k$.
We are now able to introduce the innovations model:

$$\begin{cases} \tilde{x}_{k+1} = F\tilde{x}_k + T\tilde{w}_{k+1} \\ y_{k+1} = H\tilde{x}_k + \tilde{w}_{k+1} \end{cases} \tag{8.15}$$

where

$$T = E(x_k\tilde{w}_k^T)\tilde{R}^{-1} \tag{8.16}$$

From (8.15), we get the following model too :

$$\begin{cases} \tilde{x}_{k+1} = F\tilde{x}_k + T\tilde{R}^{1/2}\tilde{u}_{k+1} \\ y_{k+1} = H\tilde{x}_k + \tilde{R}^{1/2}\tilde{u}_{k+1} \end{cases} \tag{8.17}$$

which is of the desired type (8.5). The transfer function matrix $\tilde{\Gamma}(z)$ of the model (8.17) writes :

$$\tilde{\Gamma}(z) = [I + Hz(I - Fz)^{-1}T]\tilde{R}^{1/2} \tag{8.18}$$

and is a maximal factorization of the spectral density of y_k, known as the *minimum-delay factorization* . One consequence of this fact is that the innovation may be calculated as :

$$\tilde{w}_k = y_{k+1} - H\tilde{x}_k \tag{8.19}$$

The importance of the filter model lies in that *all the minimal markovian representations of y_k have the same filter*, so that the problem we are faced with is to find this filter, given the statistics of y_k. For this reason of uniqueness, we will say that $\tilde{x}_k$ is *the* filter of y_k.

8.3 Computing the solution

We assume now that we could get in some way the covariance sequence of the process y_k. The models we consider being minimal with dimension q, it is a well-known fact in automatic control that the observability matrix :

$$O = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{q-1} \end{bmatrix} \tag{8.20}$$

has its rank equal to q and is defined up to a similarity transformation; thus the pair (H, F) is unique in the same sense. For evident reasons G, defined in (8.6), shares the same property. Thus we can conclude that the triple (H, F, G) is unique up to a similarity transformation. It may be easily obtained from the empirical covariance function $\{R_k\}$ with algorithms such that Ho's [12] or Rissanen's [26]. It is this point that we shall investigate now.

8.3.1 Estimation of the matrices H F G

Numerous algorithms exist for solving this problem and we shall focus on just one of them: the so-called *Principal Hankel Component* (PHC) [17] approximation method, which is singular value decomposition (SVD) based . It is as follows: from the covariance sequence, form the Hankel matrix:

$$\mathcal{H}_{\mathcal{M},\mathcal{N}} = \begin{bmatrix} R_0 & R_1 & R_2 & \dots & R_{N-1} \\ R_1 & R_2 & R_3 & \dots & R_N \\ R_2 & R_3 & R_4 & \dots & R_{N+1} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{M-1} & R_M & R_{M+1} & \dots & R_{M+N-2} \end{bmatrix} \tag{8.21}$$

The reason for the wide use of this matrix is the Kronecker's theorem which says that its rank is theoretically equal to the order of the associated system, i.e. equal to the dimension of state-vectors of minimal state-space realizations. Defining the matrix C as :

$$C = \begin{bmatrix} G & FG & \dots & F^{q-1}G \end{bmatrix} \tag{8.22}$$

we have that :

$$\mathcal{H}_{\mathcal{M},\mathcal{N}} = \mathcal{O}\mathcal{C} \tag{8.23}$$

Now, from the observability matrix O, define the two following matrices:

$$O^{'} = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{q-2} \end{bmatrix} \tag{8.24}$$

and

$$O^{\uparrow} = \begin{bmatrix} HF \\ \vdots \\ HF^{q-1} \end{bmatrix} \tag{8.25}$$

It is straightforward that:

$$O^{\uparrow} = O^{'}F \tag{8.26}$$

so that the matrix F is obtained as the least-squares solution of (8.26). H is obtained as the first bloc-row of O and G as the first bloc-column of C : this is the PHC approximation method.

Numerically, the factorization (8.23) is done via singular-value decomposition:

$$\mathcal{H}_{\mathcal{M},\mathcal{N}} = \mathcal{U}\mathcal{S}\mathcal{V}^T \tag{8.27}$$

O and C are obtained as :

$$O = US^{1/2} \qquad C = S^{1/2}V^T \tag{8.28}$$

The `phc` macro This macro implements the preceding algorithm to find the triple (H, F, G). In the following example, a 64-point length covariance sequence has been generated for a two-dimensional process, the first component of which is the sum of two sinusoids with discrete frequencies $\pi/10$ and $2\pi/10$, while the second component is the sum of two sinusoids with frequencies $\pi/10$ and $1.9\pi/10$, both being in additive, gaussian white noise. This is done as follows:

Then the Hankel matrix $\mathcal{H}_{\mathcal{M},\mathcal{N}}$ is built with the function `hank`. Finally, the Scilab function `phc` gives the desired triple (H, F, G).

8.3.2 computation of the filter

Now, we have obtained the triple (H, F, G) and we shall investigate the matrices T and $\tilde{R}$ that have still to be computed to completely determine the filter model (8.17). From this model, let us compute the convenient covariances:

$$R_0 = H\tilde{P}H^T + \tilde{R} \tag{8.29}$$

$$G = F\tilde{P}H^T + T\tilde{R} \tag{8.30}$$

$$\tilde{P} = F\tilde{P}F^T + T\tilde{R}T^T \tag{8.31}$$

from which the following relations hold:

$$\tilde{R} = R_0 - H\tilde{P}H^T \tag{8.32}$$

$$T = (G - F\tilde{P}H^T)(R_0 - H\tilde{P}H^T)^{-1} \tag{8.33}$$

Noting that $\tilde{R}$ and T depend solely on $\tilde{P}$ and supposing that $\tilde{R}$ is positive, we can write after elimination of $\tilde{R}$ between (8.29), (8.30) and (8.31):

$$\tilde{P} = F\tilde{P}F^T + (G - F\tilde{P}H^T)(R_0 - H\tilde{P}H^T)^{-1}(G^T - H\tilde{P}F^T) \tag{8.34}$$

which is the well-known *algebraic Riccati equation* . A matrix $\tilde{P}$ is called a solution of the Riccati equation if it is positive definite, such that $R_0 - H\tilde{P}H^T > 0$ and satisfies equation (8.34). Although this equation has several solutions, the following result gives an answer to our problem: *the covariance* $\tilde{P}$ of the filter is the minimal solution of the algebraic Riccati equation. We shall give now two algorithms giving this minimal solution : the Faurre algorithm and the Lindquist algorithm .

The Faurre algorithm [9]: in this method, the solution $\tilde{P}$ is obtained as the growing limit of the sequence of matrices P_N such that:

$$P_{N+1} = FP_NF^T + (G - FP_NH^T)(R_0 - HP_NH^T)^{-1}(G^T - HP_NF^T) \tag{8.35}$$

with the initial condition:

$$P_0 = GR_0^{-1}G^T \tag{8.36}$$

Setting $P_0 = 0$ is allowed too for this leads to $P_1 = GR_0^{-1}G^T$ hence to a simple delay in the iterations. To conclude, having obtained $\tilde{P}$ via the recursion (8.35), the matrices $\tilde{R}$ and T are computed with equations (8.32) and (8.33) respectively. This is done with the macro `srfaur`. The recursion for P_N is not implemented as in equation (8.35) for it may not converge, especially when the matrix F has poles near the unit circle. To overcome this difficulty, a factored form, Chandrasekhar type recursion has been implemented, leading to the sought after solution even in the precedent defavourable situation[1]. The Scilab function `srfaur` implements this sqare-root algorithm.

Finally, the filter and the corresponding estimated output are calculated with the help of the model (8.17) by using the macro `ltitr` which simulates linear systems. To summarize, the preceding example has been generated the following way:

The Lindquist algorithm [9]: This algorithm makes use of the fact that $\tilde{R}$ and T may be computed from $K = \tilde{P}H^T$ instead of $\tilde{P}$ itself, leading to substantial computational savings when the observation has a much lower dimension that the state, the most frequent situation. Refering the reader to [9] for the derivations, we give now the algorithm itself:

$$\left\{\begin{array}{l} K_{N+1} = K_N + \Gamma_N\tilde{R}_N^{-1}\Gamma_N^TH^T \\ \Gamma_{N+1} = [F - (G - FK_N)(R_0 - HK_N)^{-1}H]\Gamma_N \\ \tilde{R}_{N+1} = \tilde{R}_N - \Gamma_N^TH^T(R_0 - HK_N)^{-1}H\Gamma_N \end{array}\right. \tag{8.37}$$

with the initial conditions:

$$K_0 = 0 \quad \Gamma_0 = G \quad \tilde{R}_0 = R - 0 \tag{8.38}$$

Then the expression for T is the following:

$$T = (G - FK)(R_0 - HK)^{-1} \tag{8.39}$$

and `lindquist` is the corresponding Scilab function.

8.4 Levinson filtering

We still consider here y_k, a stationary, vector-valued time-series , from which we have available a sample $y_{k-1}, y_{k-2}, \ldots, y_{k-N}$; the scope here is to estimate y_k from this sample, by some $\hat{y}_k$, say, which may be written then as a linear combination of the y_{k-j}, $j = 1, \ldots, N$: $\hat{y}_k = \sum_{j=1}^N A_j^N y_{k-j}$.

[1] A mistake was found in the initialization of the algorithm and could not be corrected in time for the present document

As mentioned in the introduction, this problem may be stated as a stochastic realization problem: attempting to minimize the mean-square prediction error of order N at time k:

$$E(\|e_k(N)\|^2) = E\{\|y_k - \sum_{j=1}^{N} A_j^N y_{k-j}\|^2\} \tag{8.40}$$

the filter coefficients A_j^N, where the superscript N indicates the dependence on the sample length, are found to be the solutions of the following Toëplitz system :

$$\begin{bmatrix} R_0 & R_1 & R_2 & \dots & R_{N-1} \\ R_1^T & R_0 & R_1 & \dots & R_{N-2} \\ R_2^T & R_1^T & R_0 & \dots & R_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R_{N-1}^T & R_{N-2}^T & R_{N-3}^T & \dots & R_0 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ A_N \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_N \end{bmatrix} \tag{8.41}$$

The mean-square error Σ_N is easily calculated:

$$\begin{aligned} \Sigma_N &= E((y_k - \hat{y}_k)(y_k - \hat{y}_k)^T) && (8.42) \\ &= E((y_k - \hat{y}_k)y_k^T) && (8.43) \\ &= E(y_k y_k^T) - \sum_{j=1}^{N} A_j^N E(y_{k-j} y_k^T) \\ &= R_0 - \sum_{j=1}^{N} A_j^N R_{-j} && (8.44) \end{aligned}$$

where the second equality holds from the orthogonality principle between the estimate and the prediction error. Classically, Σ_N is taken as a measure of the quality of the estimation, since it is a monotone decreasing function of N (the longer the sample, the better the estimation). One may for example compare it to a preassigned mean-square estimation error. So, the problem is to find a procedure which calculates, successively for each N, the coefficients A_j^N and Σ_N. An answer is given with Levinson-type algorithms, an important property of which is their recursivity in the order: one particularity, relative to other input-output representations such as state-space models or rational matrices, is that in this approach the filter structure - its order for example- is considered as a parameter as well as any other, leading to the idea of lattice filter and to cascade-type realizations of transfer functions. Let us describe now the Levinson algorithm in the case of a vector-valued time-series, noting that in the scalar case the backward prediction error is no use because $R_{-k} = R_k$, leading to simpler derivations, albeit similar. For the exposition we shall make use of a Hilbert space approach, following the treatment given in 5.4.3.

8.4.1 The Levinson algorithm

Suppose we are given the vector-valued time-series y_k. We begin with some definitions:
$e_k(N)$ and $f_k(N)$ will denote respectively the forward and backward prediction errors of order N at time k:

$$e_k(N) = y_k - \sum_{j=1}^{N} A_j^N y_{k-j} \tag{8.45}$$

$$f_k(N) = y_{k-N} - \sum_{j=1}^{N} B_j^N y_{k-N+j} \tag{8.46}$$

with the convention that: $e_k(0) = f_k(0) = y_k$. We shall need also the following linear space: $Y_p^q = span\{y_p, \ldots, y_q\}$. In the present geometrical approach, the covariance $E(xy^T)$ of x and y will be noted $[x, y]$ (scalar product) and if $E(xy^T) = 0$, we shall write $x \perp y$ (x orthogonal to y), as well as $A \perp B$ for two orthogonal linear spaces. In view of these definitions, the following relations hold:

$$e_k(N) \in Y_{k-N}^k \quad \text{and} \quad e_k(N) \perp Y_{k-N}^{k-1} \tag{8.47}$$

$$f_k(N) \in Y_{k-N}^k \quad \text{and} \quad f_k(N) \perp Y_{k-N+1}^k \tag{8.48}$$

From (8.45), we get:

$$e_k(N+1) - e_k(N) \in Y_{k-N-1}^{k-1} \tag{8.49}$$

while $e_k(N+1) \perp Y_{k-N-1}^{k-1}$ and $e_k(N) \perp Y_{k-N}^{k-1}$ imply that:

$$e_k(N+1) - e_k(N) \perp Y_{k-N}^{k-1} \tag{8.50}$$

Recalling (8.48), relations (8.49) and (8.50) caracterize the space spanned by $f_{k-1}(N)$; hence there exists a matrix K_N such that:

$$e_k(N+1) - e_k(N) = -K_N f_{k-1}(N) \tag{8.51}$$

K_N is determined with the help of the orthogonality condition:

$$e_k(N+1) \perp y_{k-N-1} \tag{8.52}$$

this relation implies that:

$$\begin{aligned} &[e_k(N+1), y_{k-N-1}] \\ &= [e_k(N), y_{k-N-1}] \; - K_N.[f_{k-1}(N), y_{k-N-1}] \\ &= 0 \end{aligned} \tag{8.53}$$

hence giving:

$$K_N = [e_k(N), y_{k-N-1}][f_{k-1}(N), y_{k-N-1}]^{-1} \tag{8.54}$$

We recognize the second scalar product as the backward mean-square error Γ_N. Relations for the backward prediction error may be obtained the same way; they are:

$$f_k(N+1) - f_{k-1}(N) \in Y_{k-N}^k \text{ and} \perp Y_{k-N}^{k-1} \tag{8.55}$$

which caracterize the space spanned by $e_k(N)$; thus there exists a matrix K_N^* such that:

$$f_k(N+1) - f_{k-1}(N) = -K_N^* e_k(N) \tag{8.56}$$

and determined by the orthogonality condition:

$$f_k(N+1) \perp y_k \tag{8.57}$$

which leads to:

$$K_N^* = [f_{k-1}(N), y_k][e_k(N), y_k]^{-1} \tag{8.58}$$

Here too the second scalar product is seen to be the forward mean-square error Σ_N. Relations (8.51), (8.54), (8.56), (8.58) give the sought after recursions; their lattice structure may be explicited with the help of the following matrix polynomials:

$$A_N(z) = I - \sum_{j=1}^{N} A_j^N z^j \tag{8.59}$$

$$B_N(z) = z^N I - \sum_{j=1}^{N} B_j^N z^{N-j} \tag{8.60}$$

and the covariance matrices: $R_n = [y_k, y_{k-n}]$, from which K_N and K_N^* may be expressed:

$$\begin{aligned} K_N &= (R_{N+1} - \sum_{j=1}^{N} A_j^N R_{N+1-j})(R_0 - \sum_{j=1}^{N} B_j^N R_j)^{-1} \\ &= \beta_N \Gamma_N^{-1} \end{aligned} \tag{8.61}$$

$$\begin{aligned} K_N^* &= (R_{-N-1} - \sum_{j=1}^{N} B_j^N R_{-N-1+j})(R_0 - \sum_{j=1}^{N} A_j^N R_{-j})^{-1} \\ &= \alpha_N \Sigma_N^{-1} \end{aligned} \tag{8.62}$$

with evident definitions. The last recursion to be given is that for Σ_N, for which we will use the expressions of K_N (8.61) and K_N^* in (8.62), and the definition of Σ_N:

$$\Sigma_N = [e_k(N), y_k] \tag{8.63}$$

Then we can compute Σ_{N+1} as follows:

$$\begin{aligned} \Sigma_{N+1} &= [e_k(N+1), y_k] \\ &= [e_k(N) - \beta_N \Gamma_N^{-1} f_{k-1}(N), y_k] \\ &= [e_k(N), y_k] - \beta_N \Gamma_N^{-1} [f_{k-1}(N), y_k] \\ &= \Sigma_N - \beta_N \Gamma_N^{-1} \alpha_N \end{aligned} \tag{8.64}$$

In the same fashion, Γ_{N+1} will be written as:

$$\Gamma_{N+1} = \Gamma_N - \alpha_N \Sigma_N^{-1} \beta_N \tag{8.65}$$

To summarize,the algorithm is as follows:

$$\left\{\begin{array}{l}
\left[\begin{array}{c} A_{N+1}(z) \\ B_{N+1}(z) \end{array}\right] = \left[\begin{array}{cc} I & -K_N\ z \\ -K_N^* & z\ I \end{array}\right] \left[\begin{array}{c} A_N(z) \\ B_N(z) \end{array}\right] \\
K_N = \beta_N \Gamma_N^{-1} \\
K_N^* = \alpha_N \Sigma_N^{-1} \\
\Sigma_{N+1} = \Sigma_N - \beta_N \Gamma_N^{-1} \alpha_N \\
\Gamma_{N+1} = \Gamma_N - \alpha_N \Sigma_N^{-1} \beta_N \\
\alpha_N = (R_{-N-1} - \sum_{j=1}^{N} B_j^N R_{-N-1+j}) \\
\beta_N = (R_{N+1} - \sum_{j=1}^{N} A_j^N R_{N+1-j})
\end{array}\right. \tag{8.66}$$

with the initial conditions:

$$\left\{\begin{array}{l}
\alpha_0 = R_{-N-1}; \beta_0 = R_{N+1} \\
\Sigma_0 = \Gamma_0 = R_0; A_0(z) = B_0(z) = I
\end{array}\right. \tag{8.67}$$

The corresponding Scilab function is `levin`.

Chapter 9

Time-Frequency representations of signals

Numerous tools such as DFT, periodogram, maximum entropy method, have been presented in previous chapters for the spectral analysis of stationary processes. But, in many practical situations (speech, acoustics, biomedicine applications, ...), this assumption of stationarity fails to be true. Specific tools are then to be used when the spectral content of a signal under consideration is time dependent. Two such tools will be presented in this chapter: The *Wigner-Ville* representation and the classical *Short-Time periodogram* , which are particuliar cases of a more general class of spectral estimators[18]. Nevertheless, due to the superiority of the so-called *wigner spectrum*, no numerical computation will be done for the short-time periodogram.

9.0.2 The Wigner distribution

Let be given a discrete-time signal $f(n)$ and is Fourier transform:

$$F(\nu) = \sum_{n \in \mathbb{Z}} f(n) e^{-jn\nu} \tag{9.1}$$

The Wigner distribution of $f(n)$ is given by:

$$W_f(n, \nu) = 2 \sum_{k \in \mathbb{Z}} e^{-j2k\nu} f(n+k) f^*(n-k) \tag{9.2}$$

Useful in a time-frequency representation framework, a similar expression may be written for $F(\nu)$:

$$W_F(\nu, n) = \frac{1}{\pi} \int_{-\pi}^{\pi} e^{j2n\zeta} F(\nu + \zeta) F^*(\nu - \zeta) d\zeta \tag{9.3}$$

so that:

$$W_F(\nu, n) = W_f(n, \nu) \tag{9.4}$$

illustrating the symmetry between the definitions in both time and frequency domains. Among the properties listed in [6] are the π-periodicity and the hermitic property, the last leading to the fact that the Wigner distribution of real signals is real. One more important result is the following:

$$\frac{1}{2\pi} \int_{-\pi/2}^{\pi/2} W_f(n, \nu) d\nu = |f(n)|^2 \tag{9.5}$$

which means that the integral over a period of the Wigner distribution is equal to the instantaneous signal power.

9.0.3 Time-frequency spectral estimation

All the properties stated in [6] for deterministic signals with finite energy are still valid for random processes with harmonizable covariance, that is those for which the covariance function $K(s,t)$ has the decomposition:

$$\begin{aligned} K(s,t) &= E(X(s)X^*(t)) \\ &= \frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} e^{i(\lambda s-\mu t)} f(\lambda,\mu)d\lambda d\mu \end{aligned} \tag{9.6}$$

The same way an harmonic analysis of stationary random processes with the Fourier transform leads to the classical spectrum, a mixed time-frequency analysis can be done with the Wigner distribution, leading to a "time-frequency spectrum", which reduces to the first with the stationary assumption. In [18], a general class of spectral estimators is considered:

$$\hat{W}(n,\nu;\phi) = \frac{1}{\pi}\sum_{k\in\mathbb{Z}}\sum_{m\in\mathbb{Z}}\int_{-\pi}^{\pi} e^{ipm}\phi(p,2k)X(n+m+k)X^*(n+m-k)e^{-i2\nu k}dp \tag{9.7}$$

where $\phi(p,2k)$ is the Fourier transform of a data window caracterizing the weighting on the products. A first choice of $\phi(p,2k)$ is:

$$\phi_{STP}(p,2k) = \frac{1}{2N-1}\sum_{j\in\mathbb{Z}} h_N(j+k)h_N^*(j-k)e^{ipj} \tag{9.8}$$

leading to the well-known *short time periodogram*:

$$STP(n,\nu) = \frac{1}{2N-1}|\sum_{j\in\mathbb{Z}} X(k)h_N(k-n)e^{-i\nu n}|^2 \tag{9.9}$$

which actually is the classical periodogram of the signal to which has been applied a sliding window; smoothed versions may be obtained for this estimator[18] If now $\phi(p,2k)$ is chosen to be:

$$\phi_{SPW}(p,2k) = |h_N(k)|^2\sum_{m\in\mathbb{Z}} g_M(m)e^{-ipm} \tag{9.10}$$

equation (9.6) particularizes in the *smoothed pseudo-wigner* spectral estimator:

$$PW(n,\nu) = 2\sum_{k\in\mathbb{Z}} e^{-i2\nu k}|h_N(k)|^2\sum_{m\in\mathbb{Z}} g_M(m)X(n+m-k)X^*(n+m-k) \tag{9.11}$$

where $h_N(k)$ and $g_M(m)$ are windows with respective length $2N-1$ and $2M-1$. One major advantage in chosing $\phi_{SPW}(n,2k)$ is that it is a separable function, so that independent smoothing can be applied in the time and frequency directions, as opposed to $\phi_{STP}(n,2k)$ which is governed by "uncertainty relations" between the time and frequency weightings. Thence, the bias, known to be introduced by weighting functions in spectral estimation, can be controlled separately for the pseudo-wigner estimator. Morever, in the case of unsmoothed pseudo-wigner ($M=1$), the bias in the time direction vanishes while always present for the short time periodogram.

Now, we can compute the wigner spectrum estimator: let $x(n)$ denote the analytical signal of the sampled realization of the process $X(t)$. Setting $M=1$ (unsmoothed estimator) and $\nu_l \overset{\text{def}}{=} \pi(l/N)$, we can write with the help of (9.10):

$$\begin{aligned} PW(n,\nu_l) &= 2\sum_{k=-N+1}^{N-1} e^{-i2k\pi(l/N)}|h_N(k)|^2 x(n+k)x^*(n-k) \qquad (9.12) \\ &= 2(2Re\{\sum_{k=0}^{N-1} e^{-i2k\pi(l/N)}|h_N(k)|^2 x(n+k)x^*(n-k)\} - |x(n)|^2) \end{aligned}$$

and this expression may be easily computed via the FFT with the Scilab function `wigner`.

Example 1 : The following example is taken from [18]: the signal is a finite duration sinusoid modulated by a parabola:

$$S(t) = \begin{cases} p(t)\sin(\frac{2\pi}{16}t + u(t-488)\pi) & 408 < t < 568 \\ 0 & 0 \leq t \leq 408\ ;\ 567 \leq t \leq 951 \end{cases}$$

$p(t)$ is the parabola taking its maximum in $t = 488$, $u(t)$ is the unit step function, h_N is the 64-point rectangular window; the time and frequency increments are respectively equal to 12 and $\pi/128$; M has been set to one (unsmoothed estimator). The signal generation and wigner spectrum computation are then as follows:

`exec('wigner1.code')` Wigner analysis. Sinusoid modulated by a parabolawigner.fig1

Bibliography

[1] M. Abramowitz, I.A. Stegun, eds., *Handbook of Mathematical Functions*, National Bureau of Standards Applied Mathematics Series, no. 55, Superintendent of Documents, U.S. Goverment Printing Office, Washington, D.C., 1964.

[2] B.D.O. Anderson and J.B. Moore, *Optimal Filtering*, Englewood Cliffs, NJ: Prenctice Hall, 1979.

[3] Gerald J. Bierman, *Factorization Methods for Discrete Sequential Estimation*, Academic Press, New York, 1977.

[4] B.C. Carlson, *Special Functions of Applied Mathematics*, Academic Press, New York, 1977.

[5] E.W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, 1966.

[6] T.A.C.M. Claasen and W.F.G. Mecklenbrauker, “The Wigner distribution: a tool for time frequency signal analysis”, *Phillips J. Res.*, vol. 35, pp. 217-250, 276-300, 372-389, 1980.

[7] F. Delebecque, C. Klimann and S. Steer, *Basile, guide d’utilisation*, Internal Report INRIA, 1987.

[8] J. Dieudonné, *Calcul Infinitésimal*, Hermann, Paris, 1980.

[9] P. Faurre, *Réalisations markoviennes de processus stationnaires* , Thèse d’état, Rapport Laboria n0 13, 1973.

[10] A. Gelb, ed., *Applied Optimal Estimation*, MIT Press, 1974.

[11] Francis B. Hildebrand, *Methods of Applied Mathematics*, Prentice Hall, Englewood Cliffs, N.J., 1965.

[12] B.L. Ho and R.E. Kalman, “Effective construction of linear state variable models from Input/Output data”, *Proc. 3rd Allerton Conference*, 1963.

[13] R. Hooke, T.A. Jeeves, “Direct Search solution of numerical and statistical problems”, Journ. Assoc. Comput. Mach., Vol 8 No 2, pp 212-229, April 1961.

[14] T. Kailath, *Lectures on Wiener and Kalman Filtering*, Springer-Verlag, New York, 1981.

[15] T. Kailath, *Linear Systems*, Prentice Hall, 1980.

[16] Steven M. Kay and Stanley L. Marple, “Spectrum Analysis - A Modern Perspective”, *Proc. IEEE*, vol. 69, no. 11, pp. 1380-1419, Nov. 1981.

[17] S. Kung, “A new identification and model reduction algorithm via singular value decompositions”, Proc. 12th Asilomar Conf. Circuits, Syst. Comput., Pacific Grove, CA, pp 705-714, Nov 1978.

[18] W. Martin and P. Flandrin, “Wigner-Ville spectral analysis of nonstationary processes”, *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-33, N0 6, Dec. 1985.

[19] C. Moler, *MATLAB User’s Guide*, Tech. Rep. CS81-1, Dept. of Computer Sci., Univ. New Mexico, August, 1982.

[20] R. Nikoukhah, *A Deterministic and Stochastic Theory of Two-Point Boundary-Value Descriptor Systems*, Ph.D. dissertation, Dept. of Elec. Eng. and Comp. Science, M.I.T., 1988.

[21] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice Hall, 1975.

[22] M. Prevosto, A. Benveniste, B. Barnouin, “La méthode des variables instrumentales en treillis; application à la modélisation des structures en vibration”, Outils et modèles mathématiques pour l’Automatique, l’Analyse des systèmes et le Traitement du Signal, Vol 2, pp 639-667, Ed. CNRS, 1982.

[23] *Programs for Digital Signal Processing*, IEEE Press John Wiley and Sons, 1979.

[24] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice Hall, 1975.

[25] L.R. Rabiner, R.W. Schafer, and C.M. Rader, “The Chirp Z-Transform Algorithm and its Applications”, *Bell Sys. Tech. J.*, vol 48, pp. 1249-1292, May 1969.

[26] J. Rissanen, “Recursive identification of linear systems”, *SIAM Journal of Control*, Vol 9, 1971.

[27] G. Ruckebusch, *Représentations markoviennes de processus gaussiens stationnaires*, Thèse 3ème cyle math. stat., Paris VI, 1975.

[28] K. Steiglitz, “Designing Short-Word recursive digital filters”, Proc. 9th Allerton Conf. on Circuit and Systems Theory, pp 778-788, Oct. 1971.

[29] G.W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[30] C.F. Van Loan, “Computing integrals involving matrix exponentials”, *IEEE Trans. Autom. Control*, vol AC-23, pp. 395-404, 1978.

List of Figures